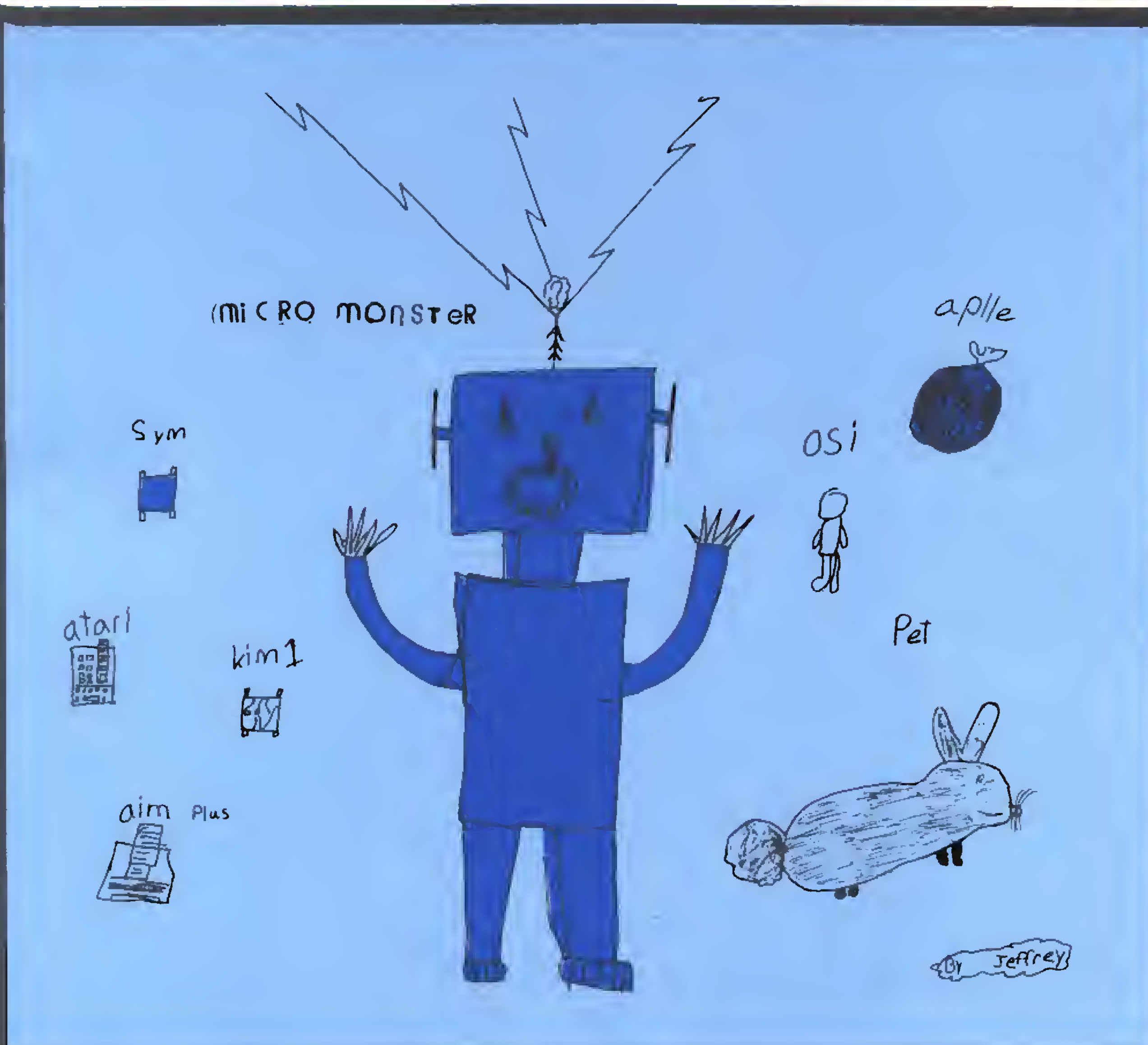
MICRO

The Magazine of the APPLE, KIM, PET
and Other 6502 Systems



NO. 19 DECEMBER 1979 $2.00

NOW PRESENTING...

Apple® software

for your Entertainment · Business · Education

Star Attractions:

FILEMASTER 2 programs: *FORMAT & RETRIEVAL* comprise a powerful data file manager. Great for everything from phone lists to legal abstracts. Needs 32K. Design your own data structure. Up to 500 characters per record. Up to 15 searchable fields in any combination. **On Disk** **$34.95**

SPACE Multi-faceted simulation of life in interstellar society. You and opponents must make life & death decisions. Keeps track of your progress from one game to next. Needs 48K and Applesoft ROM. **Disk** **$29.95**

Pot O'Gold I or our All New Pot O' Gold II A collection of 49 programs for 16K Apple. Everything from Logic to action games. Only a buck a game. Specify I or II. Price each: **Tape $49** . . . . **Disk $54**

ADVENTURE Fight off pirates and vicious dwarfs. 700 travel options, 140 locations, 64 objects. Needs ROM & 48K. **Disk.** . **$29.95**

16K CASSETTE INVENTORY Use item number, description, stock amount, reorder amount, restock date, cost & sell price. Holds up to 140 items. **Tape** **$35**

32K DISK INVENTORY: Use stock numbers description, vendor, record of purchase and sales date, amount on hand, cost & sell price, total value. Holds up to 300 items. **Disk** **$40**
With Parts Explosion: Disk **$50**

32K DATA BASE Cross file for phone lists, bibliographies, recipes. Run up to 9 lines of 40 columns each. Search by item anywhere. **Disk.** **$20**

24K HI-RES LIFE SIMULATION Conway's equations on 296x180 screen. A mathematical simulation to demo population growth with birth, death and survival as factors. **Tape** **$10**

16K CIRCUIT LOGIC DEVELOPMENT AID Evaluate circuits of up to 255 gates, including AND, OR, NOR, NAND, XOR, XNOR and INVERTER. **Tape** **$10**

16K MORSE CODE TRAINER Learn Morse Code, and transmit or receive over radio. **Tape** **$10**

16K DEVIL'S DUNGEON: Adventure through dark passages where monsters, demons, poisonous gas, dropoffs threaten . . . all to discover fantastic treasures. Comes with instruction book. **Tape** . . . **$10**

16K PACIFICA: Discover the floating island and rescue the beautiful princess. To win you must recover the enchanted crown, but you face the threat of magic spells and demons. **Tape** **$9.95**

Don't see what you've been looking for, here? Then write for our FREE SOFTWARE CATALOG. We're saving one just for you!

To order, add $2 shipping. California residents add 6% sales tax. Sorry, we can not ship to P.O. Boxes. VISA/ MASTERCHARGE and BANKAMERICARD Welcomed!

RAINBOW'S CASINO 9 gambling games: Roulette, Blackjack, Craps, Horserace, and a few originals that Vegas hasn't heard about. Needs 16K. **Tape** **$29.95**

16K SPACE WAR: You in your space capsule battle against the computer's saucer . . . in hi-res graphics. **Tape** **$12**

16K MEMORY VERIFY Diagnostic routine to check range of memory. Indicates faulty addresses, data in memory cell, and faulty data. **Tape** **$5**

16K APPLEODION Music synthesis composes original Irish jigs. Enter your own music and save on tape or disk. Includes 3 Bach fugues. **Tape.** **$10**

16K APPLEVISION Demo for Hi-Res graphics and music. **Tape** **$10**

32K COMPU-READ 5 programs to teach you speed reading, in stages. Includes synonym and antonym identification. You control your rate of speed, or keep up with the computer's pace. **Disk.** **$24.95**

48K PERCEPTION I, II, III random shapes and sizes must be matched. In III, you control format and display time and get weighted scores. Needs ROM. **Each Disk.** **$24.95**

32K STORY TELLER Use your bizarre imagination and input key words for fantastic and funny tales. Never the same story twice. **Tape** **$12.95**

32K WAR/RESCUE Engage in 10 battles with your infantry against the Apple robots. Calculate Apple's strategy and win more battles than the computer. **Tape.** **$12.95**

24K POLAR PLOT Plot polar equations in Hi-Res Graphics. **Tape** **$10**

32K SHAPE SCALER Utility to generate and animate Hi-Res graphic shapes. Simple routine provided to inspect position of shapes, and specify precise X/Y coordinates and scale. Needs ROM. **Disk.** **$13.95**

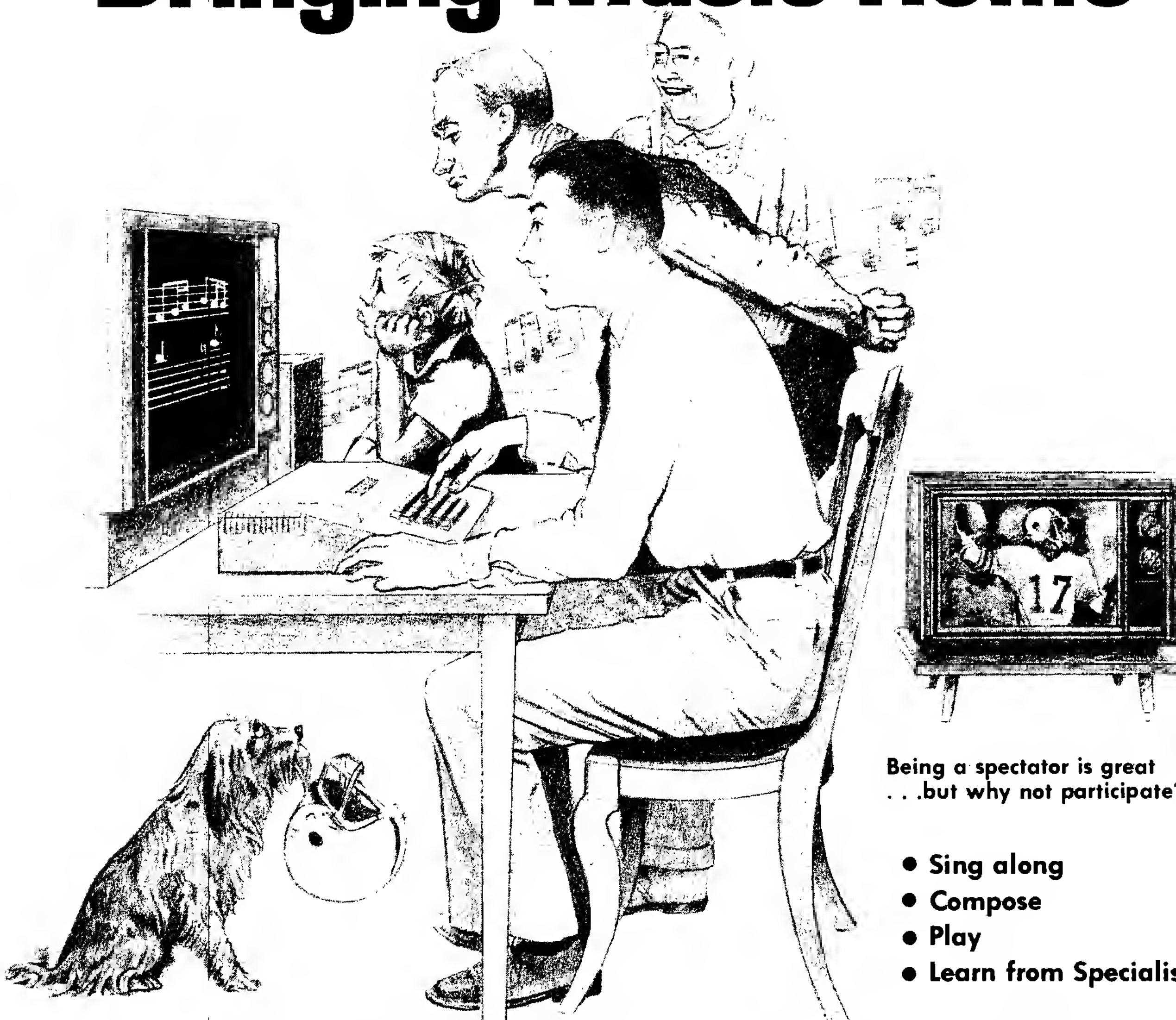
32K ZINTAR/PROPHET Great party game. Under control of the mighty Zintar's edict you take a very special trip to the world of Krintar. Heightened visual graphics. **Needs ROM.Disk** **$16.95**

APPLE MONITOR PEELED Everything you wanted to know about the Apple Monitor but couldn't figure out. User-written manual in plain English clears your confusion. **Only** **$9.95**

RAINBOW COMPUTING INC.

Garden Plaza Shopping Center, Dept. 11A
9719 Reseda Blvd., Northridge, Ca 91324
Telephone: (213) 349-5560

Bringing Music Home



Being a spectator is great
. . .but why not participate?

- Sing along
- Compose
- Play
- Learn from Specialists

LET MICRO MUSIC TURN YOUR APPLE II ® INTO A FAMILY MUSIC CENTER!

VISIT THE APPLE DEALER NEAREST YOU AND ASK FOR A DEMONSTRATION OF MMI'S MICRO COMPOSER™

The **MICRO COMPOSER LETS YOU—**

- **Play** up to **4 simultaneous voices**
- **See all 4 voices** at the same time you're **hearing** the music—a must for music editing!
- **Enter** music notes by a fast, simple and well-tested coding system.
- **Program** the **pitch, rhythm**, and **timbre of the music. Tempo** is varied by the Apple paddle.
- **Choose 7 different tone colors** for each voice or **create your own tone color.**
- **Compose, edit, display,** and **play** music through an interactive, command-driven language that's easy to learn.
- **Save** your **music** on disk or cassette.
- **Hear** quality music sound at low cost through the **MICRO MUSIC™ DAC** card. No amplifier needed! Designed for MMI by Hal Chamberlin and Micro Technology Unlimited.
- **Select** from future MMI music instruction software to accompany the **MICRO MUSIC DAC.**

Ask your local dealer for information on MMI products, or contact:

The **MICRO COMPOSER** is an **APPLE II®** compatibile, low-cost music system designed by the folks at MMI. Our music software was designed by leading experts in music education. A simple step-by-step instruction manual leads you through entering, displaying, editing, and playing music with up to four voices—soprano, alto, tenor, and bass. You can change the sound of each voice to reed, brass, string, or organ sounds and you can even color your own music sounds!

HAVE FUN! THE MICRO COMPOSER comes complete with an instruction manual, software disk or cassette—in either Integer or Applesoft **ROM BASIC,** and the **MICRO MUSIC DAC** music card. Just plug the **MICRO MUSIC DAC** into the **APPLE** extension slot and connect the audio cable to a speaker.

Suggested retail price $220.

TM Micro Music Inc 309 Beaufort, University Plaza, Normal, IL 61761

APPLE II is a trademark of Apple Computer Inc.

MICRO™

Table of Contents

December 1979
Issue Number 19

Staff

Editor/Publisher
Robert M. Tripp

Assistant Editors
Mary Ann Curtis
Evelyn M. Heinrich

Business Manager
Maggie E. Fisher

Circulation Manager
Carol A. Stark

Production Assistant
L. Catherine Bland

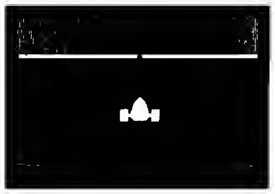
Comptroller
Donna M. Tripp

Cover Artist
Jeffrey S. Tripp
(8 years old)

Advertiser's Index

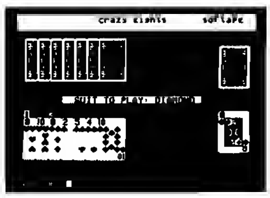
What's NEW from SOFTAPE

RACER



Slip behind the wheel, ignite the engine and get ready for a high speed race. RACER uses Hires and paddles to simulate Grand Prix excitement. Requires 24K.

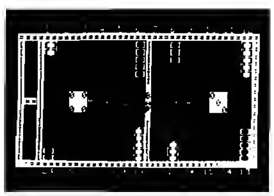
CRAZY 8's



Crazy 8's is a card game using Bill Depew's HIRES playing cards. One player can play the APPLE. The beginner can select the option of seeing the APPLE's hand while playing.

Crazy 8's is an easy to learn card game. Great for all ages. Requires 24K.

MICROGAMMON 1.0



Pit your mental skill and luck against that of the Apple with this computer implemented version of the popular board game Backgammon. All the moves are displayed on the video screen along with the board layout and pieces.

This program requires at least 16K of memory to run from cassette and 32K of memory to be stored and played from an Apple II Disk System. No additional handware is needed.

Learn, practice, and enhance your Backgammon ability a true competitor. (To our knowledge, the Apple doesn't cheat!!!)

SOLITAIRE POKER

The ultimate poker machine! SOLITARE POKER simulates the poker machines that line the Las Vegas strip. Practice your poker ability with Hires playing cards. SOLITARE POKER is a sure winner! Requires 24K.

Two More By Steve Baker

GOMOKU

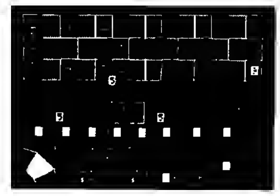
The ancient game of five men in a row. You play against a machine language routine with three levels of excellence. A Hires board using SCREEN MACHINE gives this game the beauty and style of chess. Requires 16K.

FIGHTER PILOT

It's war, and your mother ship is under attack. The adrenelin flows as you accelerate through the launch tube and penetrate the void of space. With all systems operating, your sensors show the direction of the enemy racing to meet you. After a few bursts he explodes, and you fly through his debris to meet the next one.

FIGHTER PILOT is a fast-moving game of excitement and skill. This graphics program, written in integer basic, requires 16K of memory.

ROULETTE



Roulette is a realistic duplication of the popular casino favorite using HIRES graphics and a spinning wheel. Bets can be placed with the keyboard or you can use SOFTAPE's BRIGHT Pen. One or two players can bet against the house. Requires 24K.

CRAPS

Play Las Vegas Craps on a high resolution playing table created by your APPLE. Place bets, play the field, passline and hardway rolls . . . all are possible with this detailed simulation. Dice rolls are animated on the screen. Improve your game or devise your own "system". Craps requires INTEGER BASIC and 24K or memory. BONUS!! Included on the back side of the tape is Bright Pen™ Craps for those lucky guys with the SOFTAPE Bright Pen. You will be amazed how easily moves are made and how fast the game progresses!!

PRO GOLF

Now, even on rainy days, you can improve your game with PRO GOLF. With the Apple II as your caddy, you choose your own clubs and irons on each shot on this full 18—hole course. Every fairway has its own challenging sandtraps and water hazards, but distractions disappear when the screen displays only the green as you begin to putt. Your Apple—caddy keeps track of your score. Have fun, and remember . . . keep you eye on the ball!

SOFTAPE
10432 BURBANK BLVD. • NORTH HOLLYWOOD, CA 91601

Yes! I own an Apple and I would like to receive future product announcements.

BankAmeriCard, Visa, MasterCharge & personal checks accepted.

Name ______
Address ______
City ______
State ______ Zip ______

Craps	14.95	
Racer	12.95	
S. Poker	12.95	
Crazy 8's	12.95	
Pro Golf	12.95	
Roulette	14.95	
Microgammon	14.95	
Fighter Pilot	12.95	
Gomoku	14.95	
SUB TOTAL		
Calif. Sales Tax (Cal. Res. only)		
TOTAL:		

☐ Master Charge
☐ Visa
☐ Bank Americard

Credit Card Number

INTERBANK NO. FOR MASTERCHARGE

SIGNATURE

Expiration Date MONTH YR.

VISA

master charge

Data Statement Generator

Virginia Lee Brady
D-3 Arthur Ct., Apt. 453
Salisbury, MD 21801

If you have ever had trouble getting those pesky DATA statements at the end of your BASIC program correct, then you will appreciate this program which "writes" its own DATA statements! Written for APPLESOFT, it should be adaptable to other BASICs.

I had just finished adding several new data statements to a sewing program of mine that utilized a number of data statements, and now I was reading the information into their respective arrays. "BEEP," said the Apple, "***SYNTAX ERROR." I found the offending line; I'd left out one of the elements and Applesoft would not accept "RED" as a value for "YARDS." I entered the line again and this time I typed the wrong line number and erased my previous line. There ought to be a way, I decided, to let the Apple keep track of these things. I experimented with input statements, and while these allowed me to update the arrays, I couldn't save the information.

Using the information from Jim Butterfield's article on "Pet Basic" and the information in the Applesoft Manual, I developed a program that "writes" its own data statements. This routine automatically increments the line numbers and inputs the data elements in response to appropriate prompts. It's all poked into place and becomes a permanent part of the program.

It is first necessary to understand how ROM Applesoft is stored. The basic program begins at $801 (2049 decimal) and there are only two bytes between the end of the program and the start of the simple variable table which begins at LOMEM:. Anytime a Basic line is entered, altered, or deleted, the value of LOMEM: is changed and the program must be rerun to incorporate this new value. Therefore, LOMEM: must be set at some value past the end of the program to allow for expansion of the program without writing on top of the variable table.

To use this routine it is also necessary to recognize the following locations of a data statement in Applesoft:

2 bytes—pointer to next line of Basic (to next pointer)
2 bytes—hex equivalent of the line number
1 byte—"83"—token for "DATA"
N bytes—ASCII equivalents of the program line
1 byte—"00"—indicates the end of the line

Then the sequence starts again until there are two bytes of "00" in the first two positions (total of three "00" bytes in a row.)

The program uses the fact that the locations $AF.BO (175-176 decimal) hold the value of the location where the next line number would go; or put another way, two less than this is where the "pointer to next line" would go. Call this PSN (for position). Thus the values to be poked into PSN and PSN + 1 are the low and high order bytes of the hex equivalent of LINE number. Then the DATA token (131 in decimal) is placed in PSN + 2. Since this program was designed to handle several elements in one data statement, a series of strings is next input as one string array. (It could just as easily have been done as several

MICRO™ is published monthly by:
MICRO INK, Inc.
34 Chelmsford Street
Chelmsford, Massachusetts
617/256-5515

Paid Subscribers: Nov 1979 Issue
3825

Second Class postage paid at:
Chelmsford, MA 01824

Postmaster: Send address changes to:
MICRO
P.O. Box 6502
Chelmsford, MA 01824

Publication Number:
COTR 395770

Subscription in United States:
$15.00 per year/12 issues

MICRO

"INPUT A$" 's, but using an array allows you to change a string before it is poked into memory). This is handled in lines 1035-1045. If there are no further changes, then the individual strings are concatenated into one long string with commas separating the individual substrings. Next this string is poked, one ASCII value at a time, into PSN + I + 2; then the "0" is poked into the end as the terminator.

Since PSN + 1 + 3 is the start of the next line (remember the value of I was incremented one extra time in the FOR-NEXT loop), call this NUMBER, convert it into hex, and poke it into PSN-2 and PSN-1. If the program is to be continued, PSN is given the value of NUMBER + 2 and the sequence restarted. If this is to be the last entry, then place "0" into NUMBER and NUMBER + 1. All that remains is to reset the $AF.BO pointers to reflect the new value of the end of the program (NUMBER + 2). This is done in line 1085.

List the program — the new data statement is in place at the end of the program and can be read into the necessary string of numeric variables. If you want to use this program as a subroutine to an existing data program ,where you already have some data statements being read in, you could use the fact that $7B.7C gives the line from which data is being read. Then insert a statement that sets LINE equal to PEEK(123) + PEEK(124)*256.

If your program uses trailers, then have a TRAILER$ that is the same as your trailer line (eg. "0,0,0,0"). To write over this, set PSN equal to PSN-6-LEN(TRAILER$) and your first data statement will start that much earlier and replace this trailer. At the end of the program, handle this as before and poke the TRAILER$ into place... This way every time you update your program, the original trailer is "erased" and re-appended after the last data statement.

It is important to remember that the line numbers you insert this way must be greater than those of an existing program line. If not, they will be placed at the end of the program, but will not be recognized as legitimate line numbers. (If you try to erase or list it, Applesoft, not finding it between the next lower and next greater line numbers will think it does not exist.) Also, do not try to Control-C out of the program once it has started the "poking" portion, since the pointers would be incorrect at this point and Applesoft would not know where to find the end of the program.

Since I developed this routine, I have used it in another program and in both cases I have run into only one problem. When I've added lines, saved the program to tape and later tried to reload it, I got an error message even though it still listed and ran alright. This may have something to do with the header on the cassette tape which I know contains the length of the program; but I've not yet found out how to alter this. I would appreciate any information a reader could offer. This has not, however, been a problem when a disk is used. Other than that, it's worked fine and it sure beats typing:

3000 DATA RED, SOLID, 1.25,POLYESTER

3005 DATA BLUE/GREEN, STRIPE, 1, COTTON...!!

```
10   REM EXAMPLE OF A ROUTINE THAT AUTOMATICALLY WRITES
20   REM ITS OWN DATA STATEMENTS THROUGH THE USE OF INPUT STRINGS
30   REM               VIRGINIA LEE BRADY
50   HOME
60   LOMEM: 4000
70   LINE = 2000
80   GOTO 1000
90   REM CALCULATE HI/LOW BYTES
100  HI=INT(NUMBER/256):LO=(NUMBER/256-HI)*256:RETURN
1000 REM INPUT SUBSTRINGS
1010 PSN=PEEK(175)+PEEK(176)*256
1015 INPUT"INPUT THE COLOR ";F$(1)
1016 INPUT"INPUT THE PATTERN ";F$(2)
1017 INPUT"INPUT THE YARDS IN DECIMAL ";F$(3)
1018 INPUT"INPUT THE FABRIC TYPE ";F$(4)
1020 REM ALLOW CHANGES
1035 FOR I = 1 TO 4:PRINT I; TAB(5)F$(I): NEXT I
1040 INPUT"ANY CHANGES ? ";Y$: IF LEFT$(Y$,1)="N" THEN 1050
1045 INPUT"WHICH ONE ? ";W: PRINT"CHANGE PART ";W;" TO ";: INPUT
     F$(W): GOTO 1035
1050 F$="":FOR I = 1 TO 3:F$= F$ + F$(I) + ",": NEXT: F$= F$+F$(I)
1055 LINE = LINE + 5: NUMBER = LINE: GOSUB 100
1060 POKE PSN, LO: POKE PSN + 1, HI: POKE PSN + 2, 131
1065 FOR I = 1 TO LEN(F$): PONE PSN + I + 2, ASC(MID$(F$,I,I)): NEXT I
1070 POKE PSN + I + 2,0: NUMBER = PSN + I +3:GOSUB 100
1075 POKE PSN -2,LO: POKE PSN-1,HI
1080 INPUT"ADD MORE ? ";Y$: IF LEFT$(Y$,1)="Y" THEN PSN = NUMBER + 2:
     GOTO 1015
1085 POKE NUMBER,0: POKE NUMBER + 1,0: NUMBER = NUMBER + 2: GOSUB 100:
     POKE 175,LO: POKE 176,HI
1090 END
```

Figure 1: "MAP" of Two New DATA Statements being Added

Original Last Line			First Added Line			New Last Line		
POINT LOW	08	1000	PSN-2	0A	2000	PSN-2	40	1234
POINT HIGH	10	1001	PSN-1	20	2001	PSN-1	12	1235
LINE LOW	64	1002	PSN	65	2002	PSN	66	1236
LINE HIGH	00	1003	PSN+1	00	2003	PSN+1	00	1237
"DATA"	83	1004	PSN+2	83	2004	PSN+2	83	1238
data	XX	1005	PSN+3	XX	2005	PSN+3	XX	1239
	XX	1006	PSN+I+3	XX	2006	PSN+I+3	XX	123A
"END"	00	1007		XX	2007		XX	123B
NEXT LOW	00/02	1008		XX	2008		XX	123C
NEXT HIGH	00/20	1009	"END"	00	2009		XX	123D
Orig. End		100A	NEXT LOW	36	200A		XX	123E
			NEXT HIGH	12	200B	"END"	00	123F
						NEXT LOW	00	1240
						NEXT HIGH	00	1241
					(AF.B0) →	New End		1242

Note: Original Last Line NEXT LOW/HIGH change from 0000 to 2002.

AT

Women and Children Last!

I have a feeling that the real "revolutionary" part of the microcomputer revolution is just starting to take place. Of course, parts have gotten smaller and cheaper; more software is available; new high level languages are coming along; and so forth. The real significance of all of these things lies, I believe, in the fact that millions of new people are going to get involved in computers and computing. While the overwhelming majority of individuals involved in all levels of computers currently are men, the microcomputer has made access to computers available to women and children too. This growing interest was demonstrated to me recently at a computer show in Boston. A significant number of the people who stopped by the MICRO booth to ask questions or talk about systems were women and teenagers. This issue of MICRO contains the first article by a woman. We have several articles in process from the younger set. The home computer is starting to make its effect.

I am hoping that the inclusion of these two new groups of computerists is going to have a beneficial impact on computing. Many of the individuals who owned the earliest micros were men already in the computer business in one way or another. They came to microcomputing with a large set of preconceived notions. Most microcomputer programs in use today are either games or new versions of old programs. Not many really exciting new concepts, ideas, programs, techniques, languages, approaches, etc. have appeared — yet. One of the reasons has to be the self—imposed restraints of the microcomputer 'professionals'. Since they already know 'how to solve problems', they tend to use the old tools that they are used to: BASIC, index sequential access methods, etc., and may not be alert to the new possibilities that the microcomputer provides. Where are the 'innocents' willing and able to try new directions, create chaos out of order, invent new techniques?

Watching my six and eight year old children 'attack' the computer answers the question for me. They are not interested in what "Daddy knows about the computer". They just want to push and poke and find out for themselves. And my wife — she asks some pretty insightful questions when I try to explain why a program does what it does. Perhaps the concept of 'ego-less programming' really takes on meaning when you get amateurs just having fun.

If microcomputing is going to break out of the doldrums of games and inventory control, then significant numbers of new ideas and individuals are going to have to be added to the system. Perhaps 'a child *will* lead them'!

Robert M. Tripp

How to do a Shape Table Easily and *Correctly!*

John Figueras
65 Steele Road
Victor, NY 14564

The mechanism for generating shapes and characters in APPLE High Reslution Graphics is cumbersome and prone to error. A very clear explanation of the mechanism and pitfalls is presented here. But, best of all, a program is presented which permits the user to create the shapes interactively, using the Keyboard and Display.

The Problem

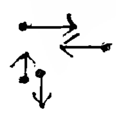
One of the most discouraging tasks facing the owner of an APPLE computer is the creation of a shape table. The table is required for generation of shapes and characters for high resolution graphics, since APPLE does not offer pre-formed plotting characters. Thus, if one wants to label the axes of a graph, the shape table can be used to supply the characters required for the labels. It is also useful for producing special shapes for games.

If, like me, the reader has ever tried to prepare a shape table using APPLE's proceedure, I am sure he/she discovered, as I did, that the proceedure is time-consumung, tedious, and error-prone. In several attempts, I have yet to generate a shape table using the manual proceedure given by APPLE, that didn't end up with missing dots, spurious projections or an unpredicted shape. At first I thought the problem was of my own making, since APPLE's directions are clear and apparently faultless. The use of the words "apparently faultless" in the last sentence implies that what I found was in fact the case: APPLE's proceedure for creating a shape table has some real glitches. I discovered these in the course of pursuing the work described below, and developed a proceedure that circumvents the glitches and produces perfect results every time. So, read on.

APPLE's proceedure for preparation of a shape table is carried out as follows: the shape is first laid out as a dot pattern on a grid (Figure 1); a series of plotting vectors is superimposed on the pattern to trace out a continuous path that covers all points to be plotted. The plotting vectors are defined either as move-only or as plot-then-move vectors.

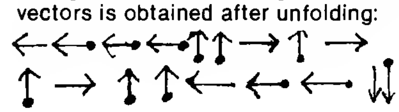
move right → move left ←
move up ↑ move down ↓

plot then move right
plot then move left
plot then move up
plot then move down



The shape in Figure 1 is reproduced in Figure 2 with the chain of plotting vectors superimposed. The plotting vector chain may start at any point, but in selecting this point you should know that the initial point in the shape is the point that gets plotted at coordinates (X,Y) in the DRAW command. Therefore, your choice of initial point determines the justification of the shape or character with respect to the plotting location. If you want a center-justified character, then start the vector sequence at the center of the shape; a left-justified character must be started at the left side, and so on. The APPLE manuals give the impression that it is immaterial where you start the shape, but if you want to have your characters fall properly on a line, it is something you must attend to. Knowing justification of the shape is important in games where things bang together and in building up large patterns by plotting sub-units adjacent to each other—cases in which it is important to know where the boundaries of the shape fall relative to the point at which it is plotted.

The next step in preparing the shape table requires that the chain of plotting vectors in Figure 2 be unfolded into a linear string, beginning with the initial point of the pattern. For the shape in Figure 2, the following sequence of vectors is obtained after unfolding:



The ploting vector string is then broken up into groups of two or three, each group (confusion!) reading from right to left. To add a little more danger to the game, the rules require that no group of vectors may end with a move-up vector or with a plot-then-move vector, in which case the group will contain at most two plotting vectors. The table in Figure 3a shows how the above string is subdivided. In this case, because of the restrictions on termination, each group can contain only two vectors. The rules for formulating these vectors groups are actually quite soundly based, as will become clear in later considerations.

We are not done yet. In the next step, each plotting vector as it appears in the table in Figure 3a is replaced by a 3-bit (octal) code. The code is shown in Figure 4, along with the decimal equivalents. Note that the decimal code for a plot-then-move vector is obtained simply by adding decimal 4 to the corresponding move-only vector. There is a certain amount of method in this madness. The 3-bit code translation for the plotting vectors in Figure 4, which represent our shape, is displayed in Figure 3b.

The next opportunity for confusion (and error) appears now, when the bit-strings in Figure 3b are re-grouped and assembled into nybbles (Figure 3c) and the nybbles are each translated into hexidecimal numbers (Figure 3d). The pairs of hexidecimal numbers, of course, represent the content of one byte. This is the byte that is stored in the shape table. In essence, then, the shape table is a list of hexidecimal numbers, which, after translation into binary and re-grouping, represents the collection of 3-bit codes equivalent to the plotting vectors, which in turn represent the original shape. In the parlance of mathematics, the shape has been *mapped* onto the set of hexidecimal numbers.

If by now the reader is feeling a tingle of impatience with this description, multiply that feeling by a factor of at least ten, and you will be on the verge of understanding what it feels like to carry out these steps. To add to the frustration, there are enough booby traps laid by APPLE to ensure quite a decent probability that after you have gone through this travail, the shape that finally appears on your screen will be misshapen. With a computer at hand, it seems silly to be bogged down by a process like this—and that's what the rest of this article is about: a computer program in APPLESOFT BASIC that allows easy graphic input of a shape or character with automatic generation and storage of a correct shape table—graphics without tears, so to speak.

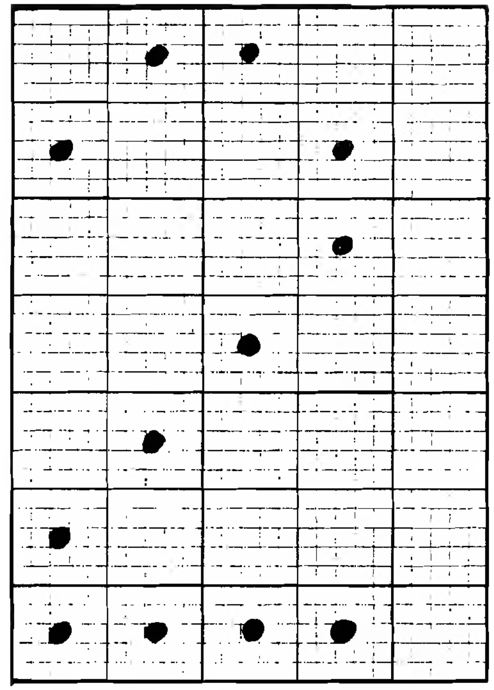


Figure 1: **Shape to be coded**

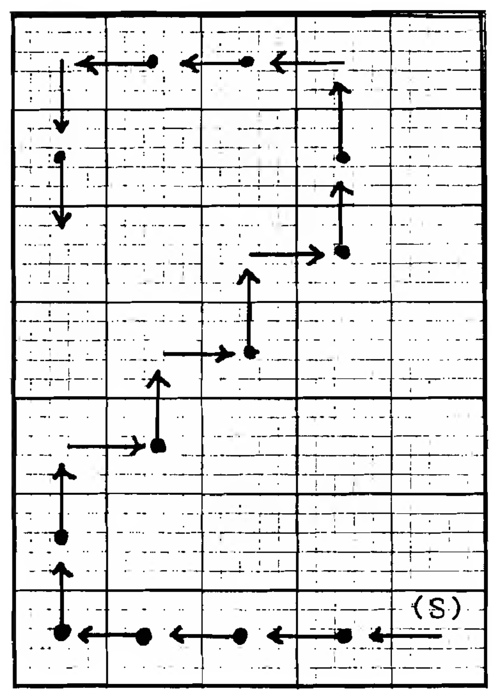
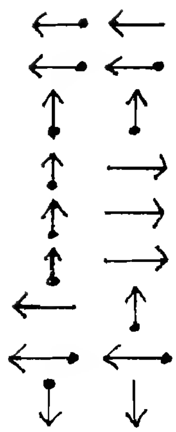


Fig. 2: **Layout of Plotting Vectors. (S) is the starting point. With this choice of (S), the shape will be lower right justified and will plot with one empty column to the right of the shape.**



(b)	(c)	(d)
00 111 011	0011 1011	3B
00 111 111	0011 1111	3F
00 100 100	0010 0100	24
00 100 001	0010 0001	21
00 100 001	0010 0001	21
00 100 001	0010 0001	21
00 011 100	0001 1100	1C
00 111 111	0011 1111	37
00 110 010	0011 0010	32

(a)

***Fig. 3:* Translation of shape vectors to Hexidecimal Code**

Approach to a Solution

Every computer programmer has his own mind-set. For some, it is structure: a beautiful program that reads like a novel. For others—start at the middle and develop a nice, tight, efficient algorithm. I am an input-output bug. To me, the proper questions that should be first answered are: how can I make it easy for the user of the program to get his data into the program; and how can the output be made digestible? In the present case, of course, the major problem is one of input. With the equipmant at hand—an APPLE keyboard, video screen and a couple of floppy disks—I settled on a display of a 15 x 15 grid and a cursor that can be moved by hitting appropriate keys (Up, Down, Left, and Right). The shape is created by plotting the shape as a dot pattern under control of the moveable cursor, using the P (for Plot) key to lay down the dot pattern. One necessary key is the Quit key, which informs the computer that the shape is done. A convenience key, E for Erase is provided to accomodate some of my sloppy keyboard habits; it facilitates undoing the last plotted point. The selection of keys U,D,L and R for directing the cursor was modeled after the set of allowed plotting vectors (there are no diagonal moves in the set), and was a fortunate selection for easy formulation of the algorithm.

While the general format for input was quite clear, the approach to translating that input into a shape table was not immediately clear. Two proceedures are possible: you can store all of the input data in some sort of two-dimensional array in memory and then analyze it, or you can take the input data as they are acquired and develop the shape table on the fly. I seriously considered the first path, and in fact, wrote a program that would translate the input pattern into a matrix of zeroes and ones. Further consideration showed that analysis of the pattern would be difficult, one of the major problems being that of ensuring proper plotting of the shape with respect to its starting point, i.e., justification. Moreover, the most efficient approach in terms of processing time and storage requirements for the shape table is to confine generation of the plotting vectors to the occupied cells of the grid as much as possible. Such pattern tracing on an arbitrary two dimensional array presents a formidable search problem, particularly with disconnected patterns. The solution of the problem of efficienly tracing the input pattern was obvious as soon as I realized that the keystrokes used by a person entering the pattern on the grid constituted a continuous record of the pattern. By analyzing the keystroke pattern, I could produce a string of equivalents. The inspiration for this may be tracable in part to my knowledge of the way in which chemical structures are recorded at Chemical Abstracts Service of the American Chemical Society, where chemical typewriters, used for creating chemical structures, are connected to computers which record the keystrokes of the operator entering the structure. The recored of keystrokes can then be "played back" to reproduce the structure exactly as it was keyed in. With this basic approach decided upon, the outline of the required algorithm became clear:

1) Select the position in memory at which the shape table is to be stored.
2) Generate and display the working (15 x 15) grid.
3) Input the starting coordinates for the shape (required for justification).
4) Generate the proper 3-bit codes that represent the plotting vectors, based on the keystrokes used to input the pattern.
5) Assemble the 3-bit codes (in groups of two or three, depending upon APPLE'S strictures) into a byte.
6) Store the assembled byte in the shape table.
7) Provide for proper finishing-off of the current byte when the Quit key is hit.
8) Add an end-of-record mark (a zero byte) required by APPLE as a shape terminator.
9) Store the table.

Most of these steps are straightfoward, but two of them, generation of the 3-bit codes that represent plotting vectors, and their assembly into bytes (steps 4 and 5, above), require further elaboration.

In APPLESOFT BASIC, the character returned by a keystroke is accessible with a "GET" command; the instruction GET KEY$ will load the character accessed by the next keystroke into the variable KEY$. We may examine KEY$ to determine whether it contains a "D", "L", "U", or "R" and then do a table look-up (using the definitions in Figure 4) to retrieve the *decimal* value associated with the direction implied by the keystroke. Each decimal value, of course, as stored in memory will generate the proper 3-bit binary code. Subsequently, the keystroke *preceding* the current one (which we thoughtfully saved in variable KSVE$) is examined. If KSVE$ is a "P", then the current 3-bit code must represent a plot-then-move vector and decimal 4 us added to the deciaml factor for the current key. If KSVE$ is not a "P", then the current decimal key equivalent remains unaltered.

Assembly of the 3-bit codes into bytes involves only basic consideration of decimal to binary conversion. Byte assembly is done in the program as each 3-bit code becomes available, but for the purposes of discussion, let us assume that 3-bit codes, V_1, V_2, V_3 are available in that order from the last three keystrokes. The first 3-bit code initializes the byte:

$$\text{BYTE} = V_1 \qquad 00000\overbrace{XXX}^{V_1}$$

The second 3-bit code must be added to the byte, but must first be left-shifted three bits if the V_1 bits already present

are to remain unchanged. This is done by multiplying V_2 by 8:

$$\text{BYTE} = \text{BYTE} + 8^*V_2 \qquad 00\overbrace{YYY}^{V_2}\overbrace{XXX}^{V_1}$$

Now for V_3. To refresh your memory, you will observe in Figure 4 that all plot-then-move 3-bit codes have their left-most bits "on." Since there are only two bits remaining unfilled in the byte, there is no way in which the plot status of the third 3-bit code can be entered into the byte. In this case, processing of the byte stops, and it is stored in the shape table, while V_3 is used to initialize the next byte. This is the reason that plotting vectors cannot be stored as end vectors in a byte, one of APPLE'S restrictions previously noted. In similar fashion, if V_3 corresponds to a move-up vector, with all bits zero, it is not loaded into the current byte, but is used to initialize the next byte. The reason for this is not so obvious, but is related to the aforementioned deduction that plotting vectors cannot appear as end vectors in the byte. For, suppose that the zero move-up vector V_3 *could* be stored as an end vector; then everytime V_3 happened to be a plotting vector, the last two bits in the byte would be a zero, and undesired up-moves would be enabled whenever a plot-then-move vector happened to occur in V_3. APPLE'S restrictions make sense!

In the event that V_3 is neither a move-up nor a plot-then move vector, it is added to the byte, for it then consists of an unambiguous two-bit code (Figure 4) that can fit into the remaining two bits of the byte. Addition of V_3 requires a 6-bit left shift of V_3 to avoid changing the bits already present. This is done by multiplying V_3 by $64(=2^6)$:

$$\text{BYTE} = \text{BYTE} + 64^*V_3 \qquad \overset{V_3}{ZZ}\overset{V_2}{YYY}\overset{V_1}{XXX}$$

Earlier, I mentioned glitches designed into APPLE'S shape procedure that would offer problems in obtaining correct shapes in graphics. There are actually two kinds of glitches, one predictable and the other not. The predictable one is a consequence of two facts: 1) APPLE uses a zero byte as an end-of-record mark to terminate every shape; 2) the move-up vector is represented by a 3-bit code of 000. It follows that several move-up vectors in a row will generate an end-of-record mark and any part of the shape following thereafter will be forgotten. That's bad enough. Worse is the unexpected fact that move-up codes (000) that lie on the left part of the byte (most significant bits) are not recognized. For example, consider the two cases of a plot-then-move right command followed by a move-up command,

00000101 (decimal 5)

and a move-up command followed by a plot-then-move right command,

00101000 (decimal 40).

Presumably, these commands should give the same net result. That's what you think, and what I thought also! In fact, the move-up command implied in the left bits of decimal 5 is not recognized by the system, and the byte is interpreted as a plot-then-move right instruction only. Therefore, if you try to generate a 45° line with the sequence

plot-then-move-right: move-up: plot-then-move-right: move-up...

you will get a horizontal line, whereas the sequence

move-up: plot-then-move-right: move-up: plot-then-move-right...

will give the desired 45° line!! There is nothing in APPLE'S literature that would lead the unwary to suspect that these two sequences will not plot alike. Now you know the source of those misshapen shapes.

The two problems described in the preceding paragraph-premature end-of-record mark and non-plotting up-vectors that appear in the left bits-arise from the definition of the up-vector as a zero 3-bit string. In fact, a concise statement of the problem is that any byte with a value less than decimal 8 can be expected to misbehave, unless it is the last byte in the shape table. The solution to the problem lies in preventing the occurence of these dubious bytes. This can be done easily-especially with a computer program-by introducing dummy right- and left-moves. The technique is simple: check the value of the assembled byte; if it is less than decimal 8, the second vector in the byte must correspond to the move-up (000) vector. In that case, replace the left-most zero bits by a non-zero, move-right vector, transfer the move-up (000) vector to the *next* byte and follow it by a move-left vector. By placing the move-up (000) vector into the right-most three bits of the next byte, you ensure that it will be recognized as an up-vector. The succeeding move-left vector un-does the effect of the move-right vector installed in the preceeding byte so that the correct shape is maintained. Implementation of this routine in a computer program is actually quite easy, and resolves the problems introduced by the up-vector. Frankly, I don't see how anyone could be expected to obtain predictable shapes from APPLE'S procedure using hand-methods for creating shape tables, considering the inherent problems posed by the zero up-vector.

THE PROGRAM(S)

Three programs were written to implement the computer-guided formulation of a shape table: A) a shape file initialization program (Figure 5); B) a shape creating program (Figure 7); C) a shape display program (Figure 8). These will be discussed briefly. I hope that the folowing discussions coupled with the comments scattered through the programs will enable you to follow the programs without difficulty.

Plotting Vectors	3-bit Codes	Decimal Equivalents
↑	000	0
→	001	1
↓	010	2
←	011	3
•↑	100	4
•→	101	5
•↓	110	6
←•	111	7

Fig. 4: **Representation of Plotting Vectors as 3-bit Codes and decimal equivalents**

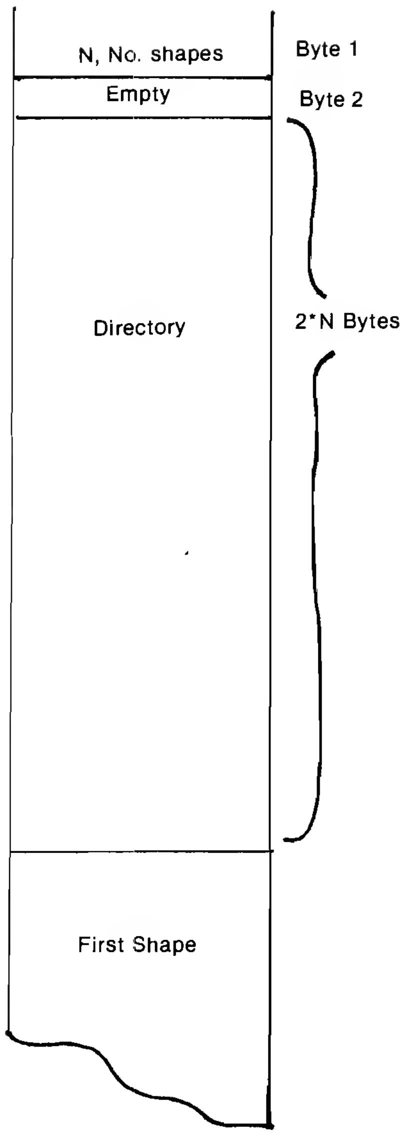


***Fig. 5:* Memory Map for Shape Table**

Shape File Initialization

The principle shape-creating program requires a previously allocated disk file for shape table storage. The initialization program (Figure 6) creates the disk file and also establishes the name and length of the file. The program allocates space for the shape table directory based on the number of shapes to be stored in the file, a number that is declared by you during initialization. The memory map for a shape table is stored in the first byte of the table; its maximum value is therefore 255, and this is the maximum number of shapes that can be stored in one shape table. The directory contains addressing information that allows random access to any shape in the table.

The directory falls between the first byte of the table and the beginning of the first shape. The amount of space allocated to the directory is determined by the number of shapes ultimately to be stored in the table; each shape requires two byte in the directory for addressing. The shape tables themselves may be any length, up to a total length consistent with the 15 x 15 matrix in which the shapes are created. The shape tables are stored end-to-end as they are added to the file, each shape determining in a zero byte as end-of-record mark. The layout of the shape file requires that any tables added to the file be accurately done, because once a table is buried in the file, it cannot be simply replaced unless the replacement has precisely the same length.

The file initialization program is also used for creating the cursor required for mapping shapes on the 15 x 15 working grid produced by the principal program. This relieves the user of the need to generate the cursor himself everytime he opens a new shape file. The cursor is stored as the first shape in the shape file, and the shape-creating program assumes that the cursor has already been stored for its use. As a consequence of this arrangement, you must remember that the user-generated shapes start with the *second* shape table in the file.

Although the file initialization program zeroes out all of the bytes in the directory, there is no substantial reason for doing this, except that the string of zero bytes make it easy to determine where the directory ends and the shape tables begin in a memory dump. This advantage will last only until the directory is filled.

The Shape Creating Program

The BASIC program (Figure 7) that enables shape generation requires the use of dual floppy disks, but can be easily changed for single floppy use by replacing "D2" in step 110 by "D1." (Similar adjustments will have to be made in the initialization and display programs, which store and access the shape file from disk D2). Tape users will have to replace disk I/O by suitable tape I/O in steps 100, 110 and 1360.

The program loads a pre-existing shape file (created by the initialization program, if necessary) from disk, using the shape file name supplied by you on request from the program. The file is loaded into a memory location which you are also asked for by the program. A check is made (step 220) that there is room in the shape file directory for another entry. If not, you will be so advised and the program will abort. A pointer to the shape file required by the APPLE system is set up in step 260. The 15 x 15 plotting grid is turned on (steps 300-330) and you will be asked to input the starting grid coordinates for the shape. Note, these are *grid* coordinates and *not* screen coordinates that are asked for. The cursor will be displayed on the center of the grid square that you have just selected as the starting point. Some user helps are displayed in the text area under the grid (steps 410-440), and you are off and running. Manipulation of the R,L,D, and U keys will move the cursor in the appropriate directions. The REPEAT key will work with these commands. Pressing the P key will plot a small circle inside the square in which the cursor currently resides, and this plotted point will become part of the shape table being built in memory. An image of the cursor will persist in the initial square—as a "negative" image if you happened to plot at that square. The persistent cursor image serves as a reminder to you of the location of the start of the shape. The cursor is made to disappear and reappear in adjacent squares as you press the move keys by XDRAW commands at steps 500 and 530; the IF statement at step 1040 in the subroutine that draws the plotting circle is responsible for keeping the persistent image of the cursor at the starting square. The flag, FLAG, that appears in step 480 and elsewhere is used to allow the cursor to be turned off in a plotted square and to be turned on again when the cursor moves to the next square.

Keystrokes are recorded in step 570. A previous step (550) saves the previous *two* keystrokes in KI$ and KSVE$. The former record, KI$, is required to allow the erase feature, controlled by the E key and discussed below. KSVE is needed for proper generation of plot-then-move 3-bit codes, also discussed below. Interpretation of a keystroke takes place in steps 590-710, a sequence of IF's called a *sieve*. This particular form of key screen was chosen because it gives almost complete protection against inadvertent entry of incorrect keys. Once you are in the program, you will find that the keyboard is effectively locked out for all keys except those required by the program. If a non-applicable key is pressed, the sieve eventually routes the program through step 710 back to another key access at step 570. Inside the sieve, when a keystroke has bee identified as a move command (L,R,U,D), the appropriate X- or Y- coordinate adjustment is made and the decimal value of the 3-bit code applicable to the move is stored where the variable KSVE$ is checked to see if the previous keystroke was a Plot command. If it were, SYMBOL is incremented by a 4 (remember Figure 4?), and SYMBOL is then transmitted to the byte assembly area, more of this later.

If the current keystroke corresponds not to a move command, but to a Plot command, the program sets the cursor disable flag, FLAG, calls the plot subroutine and then branches back to get the next keystroke (all of this is done in step 680). The Quit command forces a branch to a routine that closes out the current byte (starting at step 1080), adds a record mark (step 1170) and draws thew completed shape (step 1170). At this juncture, you are asked a series of questions, the answers to which will allow you to:

1) forget the current shape and go back and try again without re-accessing the current shape file from disk;
2) keep the current shape, update the shape file directory and start a new shape;
3) forget the whole thing—add no new shapes to the file and quit;
4)load an updated shape file to disk and quit.

These alternatives will help you to avoid filling up the shape table with unwanted shapes, and allow you to experiment without being forced to save all of your experiments.

The closing out of the current byte preparatory to ending the current shape definition (step 1080) poses a problem if the last keystroke is a Plot command because a P command alone does not generate a vector. There is nothing to store after a final P command, unless it is followed by some sort of move. The problem is handled in steps 1100-1140 by adding an arbitrary up-move after a final Plot command to generate a plot-then-move-up vector. (Note that in the illustration Figure 2, the concluding vector is a plot-then-move-down. This was done for the sake of clarity in drawing only. The point is mentioned in case some unusually perceptive reader notices that the foregoing description does not tally with the example in Figure 2). The final vector is either added to the current byte, in which it will appear as the only entry. If the last keystroke prior to closing the current shape table is anything other than a Plot command, the current byte can be closed out immediately without further ado.

The erase command has the very limited capability of erasing the last Plot command only. As discussed before, a Plot command alone does not result in formation of a vector until it is followed by a command. Therefore, if a Plot command is issued in error and no move command follows it, no vector will be generated and the shape table remains unchanged at this point. It is therefore possible to undo the Plot command simply, without the complication of analyzing the last byte for returning to the state that preceeded the mistaken command (and it would be complicated!!). At the point at which the Plot command is mistakenly issued, KSVE$ has a certain value. If we wish to go back to the condition prior to the mistaken Plot command, we must restore that value to KSVE$ so that when the correct command is issued it is properly interpreted when KSVE$ is examined subsequently. The character required for this purpose lies waiting in KI$. Thus, the erase command loads this previous value into KSVE$ and "unplots" the incorrect plotting circle by re-plotting with the color "black" (HCOLOR = 0 in step 720). Note that because of these limitations, no plot command can be undone after a move has been made.

Byte assembly using the 3-bit codes (stored currently in SYMBOL) occurs in 780-980. The variable CYCLE keeps track of the number of 3-bit codes entered into the current byte (called BYTE in the program). After the second 3-bit code is loaded into BYTE (step 820) a check is made (step 840) to see if the byte is less than 8; if it is, we know that the byte contains an unrecognizable move-up vector in the left five bits. In that case, a dummy move-right 3-bit code is inserted into the byte, the byte is stored (step 860) and a new byte is formed consisting of the required move-up (000) followed by a dummy move-left (110) to compensate for the dummy move-right. The resulting byte contains the bit string 0001 1000, decimal 24, generated in step 880. Statements 950-980 take care of the cases in which the third 3-bit code is a plot-then-move code or a move-up only code, which require that the current byte be stored, and the current 3-bit code be loaded into the next byte.

The Display Program

It is likely that your disk or tape will be replete with shape files tailored to various uses, now that creating shape tables is so easy. A convenient display program will become essential in order to find out which shapes are stored where. The display program that accomplishes this (Figure 8) is an example of how shape files may be used is a program. The program constructs a 6 x 6 grid on the high resolution screen and displays one shape per grid cell. To identify the location of the shapes in the shape table, each occupied cell carries the shape index in the upper left-hand corner. The numerals required for plotting these indices are extracted from a shape table called NUMERALS that you will have to create at storage location 20000 (decimal) by means of the shape creating program. The numerals are restricted to a 5 x 7 grid, and are formatted as illustrated by the example in Figure 1. Sufficient space is reserved in the display squares to accomodate three-digit numerals from 1 through 255. "Aha," you ask, "how can 255 shapes be displayed in a 6 x 6 grid?" The program provides for paging through the shape table, 36 shapes at a time. The paging is activated by hitting any alphanumeric key on the APPLE keyboard.

The display program opens by getting the shape files that it needs—one for numerals (step 50) and the table to be displayed (step 90). Pointers to the tables are set up (steps 70 and 120). Starting at step 180, each shape I is accessed in a FOR...NEXT loop. A grid-specific index is calculated (step 190) by taking the current shape index I modulo 36(step 190). For the first shape in each group of 36 (I modulo 36 = 1), the screen is cleared (step 240) and the 6 x 6 grid is displayed (steps 250-330). The row and column positions for the I—th shape in the grid are found (steps 360, 370). The shape index is "unpacked" into its separate digits (steps 380-410) and these digits are plotted in the correct grid cell in the upper left-hand corner (steps 430-480). The NUMERALS shape table is accessed in step 420 by placing the pointer to the NUMERALS shape table in (decimal) addresses 232 and 233, so that subsequent DRAW commands will refer to this table. In similar fashion, when the shapes to be plotted are required, the address of the shape table must be entered into addresses 232, 233. This program illustrates how any number of shape tables may be used inside a program simply by supplying the correct pointers at the time that shapes are to be DRAWn or XDRAWn.

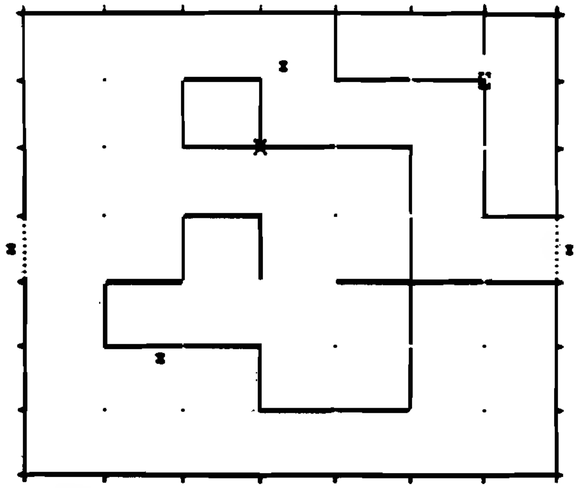
Parting Words

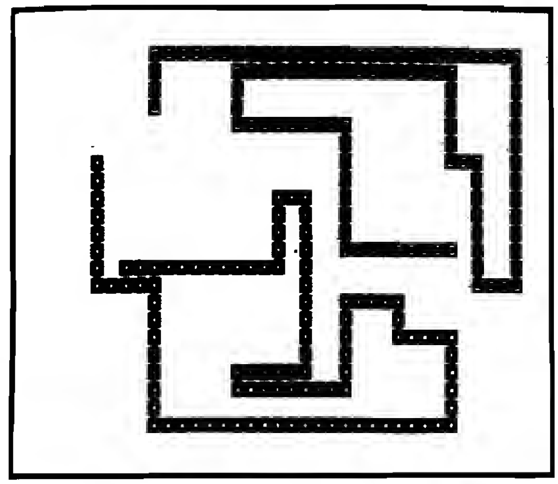
The 15 x 15 grid used for shape creation is the largest practical size for the APPLE screen with space provided for text. A larger grid can be accomodated by eliminating the text area, but this will compromise the required starting coordinate input. However, the number of cells could be increased by decreasing cell size and using a smaller plotting figure. If you try this, it is convenient to select a plotting grid with odd numbers of X and Y segments so that the central plotting area falls on a grid square and not at the intersection of two grid lines. This is of help in centering shapes.

You should also be aware, if it is not obvious by now, that the location of a shape on the grid has no bearing on where it plots in high resolution graphics, except with regard to the initial point of the shape, which alone determines justification. You may use any convenient subsection of the full grid for plotting, and it does not have to be the same subsection for each shape.

continued on page 19

Software for the Apple II





SCORE: 108   SCORE: 105

DYNAMAZE—a dazzling new real-time game. You move in a rectangular game grid, drawing or erasing walls to reflect balls into your goal (or to deflect them from your opponent's goal). Every ball in your goal is worth 100 points, but you lose a point for each unit of elapsed time and another point for each time unit you are moving. Control the speed with a game paddle: play as fast as ice hockey or as slowly and carefully as chess. Back up and replay any time you want to; it's a reversible game. By Don Stone. Integer Basic (plus machine language); 32 K; $9.95.

ULTRA BLOCKADE— the standard against which other versions have to be compared. Enjoy Blockade's superb combination of fast action (don't be the one who crashes) and strategy (the key is accessible open space—maximize yours while minimizing your opponent's). Play against another person or the computer. New high resolution graphics lets you see how you filled in an area—or use reversibility to review a game in slow motion (or at top speed, if that's your style). This is a game that you won't soon get bored with! By Don Stone. Integer Basic (plus machine language); 32 K; $9.95.

What is a REVERSIBLE GAME? You can stop the play at any point, back up and then do an "instant replay", analyzing your strategy. Or back up and resume the game at an earlier point, trying out a different strategy. Reversibility makes learning a challenging new game more fun. And helps you become a skilled player sooner.

WORLD OF ODYSSEY—a new adventure game utilizing the full power of Disk II, which enables the player to explore 353 rooms on 6 different levels full of dragons, dwarfs, orcs, goblins, gold and jewels. Applesoft II 48K; $19.95 includes diskette.

PERQUACKEY—an exciting vocabulary game which pits the player against the clock. The object of the game is to form words from a group of 10 letters which the computer chooses at random. The words must be 3 to 10 characters in length with no more than 5 words of any particular length. Each player has only 3 minutes per turn. The larger the words the higher the score. Applesoft II 16K; $9.95.

APPLESHIP—is a naval game in which two players enter their ships in respective oceans. Players take turns trying to blast their opponent's ships out of the water. The first player to destroy their opponent's ships may win the game. A great low-res graphics game. Applesoft II 32K; $14.95.

Available at your
local computer store

Call or write for our free
SOFTWARE CATALOG

Apple II is a registered
trademark of
Apple Computer, Inc.

DEALER INQUIRIES INVITED

POWERSOFT, INC.

P. O. BOX 157
PITMAN, NEW JERSEY 08071
(609) 589-5500

Programs Available on Diskette at $5.00 Additional

- Check or Money Order
- Include $1.00 for shipping and handling
- C.O.D. ($1.00 add'tl. charge)
- Master Charge and VISA orders accepted
- New Jersey residents add 5% sales tax

6: Shape File Initialization Program

```
10  REM  SHAPE FILE INTIALIZATION

20  INPUT "NAME OF SHAPE TABLE ";
NAME$
30  INPUT "STARTING ADDRESS,DECIM
AL ";ADDR
40  INPUT "NO. OF SHAPES TO BE ST
ORED ";N
50  REM  ZERO DIRECTORY
60  FOR I = 0 TO 2 * N + 1
70  POKE ADDR + I,0: NEXT
80  REM  CALCULATE INDEX TO CURSO
R
90 N = 2 * N + 2
100  REM  PUT CURSOR INDEX INTO D
IRECTORY
110  POKE ADDR + 2,N - 256 *  INT
(N / 256)
120  POKE ADDR + 3, INT (N / 256)

130  REM  CALC INITIAL ADDRESS TO
 CURSOR
140 INIT = ADDR + N
150  REM  ENTER CURSOR SHAPE VECT
ORS
160  DATA  62,36,45,54,04,00
170  FOR I = 0 TO 5
180  READ A: POKE INIT + I,A: NEXT

190  REM  GET INDEX TO NEXT SHAPE

200 N = N + 6
210  REM  STORE NEW INDEX IN DIRE
CTORY
220  POKE ADDR + 4,N - 256 *  INT
(N / 256)
230  POKE ADDR + 5, INT (N / 256)

240  REM  UPDATE SHAPE COUNTER
250  POKE ADDR,1
260  REM  STORE INITIALIZED FILE
ON DISK
270 D$ =  CHR$ (4)
280  PRINT D$;"NOMON C,I,O"
290  PRINT D$;"BSAVE" + NAME$ + "
,A" +  STR$ (ADDR) + ",L" +
 STR$ (N) + ",V0,D2"
300  END
```

7: Shape Creating Program

```
10  PRINT  TAB( 6);"****CREATE A
SHAPE TABLE****"
20  PRINT
30  PRINT  TAB( 5);"BY J. FIGUERA
S, ROCHESTER, N.Y.": PRINT
40  PRINT  TAB( 16)"9/12/79": PRINT

50  PRINT  TAB( 17)"*****": PRINT

60  REM  INPUT TABLE NAME AND LOC
ATION
70  INPUT "SHAPE TABLE NAME ";NAM
E$
80  INPUT "STARTING ADDRESS,DECIM
AL ";ASVE
90  REM  DISK ACCESSES USE DISK D
2
100 D$ =  CHR$ (4): PRINT D$;"NOM
ON C,I,O"
110  PRINT D$;"BLOAD " + NAME$ +
",A" +  STR$ (ASVE) + ",V0,D
2"
120  REM  GET CAPACITY MAX OF FIL
E
130 MAX =  PEEK (ASVE + 2) + 256 *
 PEEK (ASVE + 3)
140 MAX = (MAX - 2) / 2
150  REM  GET NO. OF SHAPES IN TA
BLE
160 N =  PEEK (ASVE)
170  REM  GET FILE LENGTH
180 INDEX =  PEEK (ASVE + 2 * N +
2) + 256 *  PEEK (ASVE + 2 *
N + 3)
190  REM  COMPUTE ADDRESS OF NEXT
 FREE BYTE
200 ADDR = ASVE + INDEX
210  REM  SEE IF FILE IS FULL
220  IF MAX > N THEN 250
230  PRINT "SHAPE TABLE FULL. NEX
T FREE BYTE AT ";ADDR
240  GOTO 1370
250  REM  SET UP APPLE POINTERS T
O TABLE
260  POKE 232,ASVE - 256 *  INT (
ASVE / 256): POKE 233, INT (
ASVE / 256)
```

```
270  REM  UPDATE SHAPE COUNTER
280 N = N + 1: POKE ASVE,N
290  REM  DISPLAY PLOTTING GRID.
INITIALIZE COUNTER, CYCLE
300  HCOLOR= 3: SCALE= 1: ROT= 0:
CYCLE = 0
310  HGR
320  FOR X = 0 TO 150 STEP 10: HPLOT
X,0 TO X,150: NEXT
330  FOR Y = 0 TO 150 STEP 10: HPLOT
0,Y TO 150,Y: NEXT
340  REM  CLEAR TEXT AND GET INIT
IAL PLOT COORDS
350  PRINT : PRINT : PRINT : PRINT

360  PRINT "ENTER STARTING COORDS
"

370  INPUT "X ";X:X = 10 * X - 5
380  INPUT "Y ";Y:Y = 10 * Y - 5
390  DRAW 1 AT X,Y:XS = X:YS = Y
400  REM  CLEAR TEXT, DISPLAY INS
TRUCTIONS
410  PRINT : PRINT : PRINT : PRINT

420  PRINT "MOVE PLOT CURSOR WITH
 KEYS"
430  PRINT "  L-LEFT  R-RIGHT  U-
UP  D-DOWN"
440  PRINT "  P TO PLOT, Q TO QUI
T."
450  REM  INITIALIZE KEY$,PLOT CU
RSOR
460 KEY$ = "":KSVE$ = "": GOTO 57
0
470  REM  FLAG RE-ENABLES CURSOR
AFTER A PLOT DISABLE
480  IF FLAG = 1 THEN 520
490  REM  ERASE CURSOR IN PREVIOU
S SQ.
500  XDRAW 1 AT X1,Y1
510  REM  PLOT CURSOR AT NEW X,Y.
 SAVE X,Y
520 X1 = X:Y1 = Y:FLAG = 0
530  XDRAW 1 AT X,Y
540  REM  SAVE LAST TWO KEYSTROKE
S. KI$ IS NEEDED FOR ERASE R
OUTINE.
550 KI$ = KSVE$:KSVE$ = KEY$
560  REM  GET NEW KEYSTROKE
570  GET KEY$
580  REM   GO TO SIEVE TO GET 3-B
IT PLOT VECTOR FROM KEY$ AND
 KSVE$
590  IF KEY$ < > "U" THEN 610
600 SYMBOL = 0:Y = Y - 10: GOTO 7
60
610  IF KEY$ < > "R" THEN 630
620 SYMBOL = 1:X = X + 10: GOTO 7
60
630  IF KEY$ < > "D" THEN 650
640 SYMBOL = 2:Y = Y + 10: GOTO 7
60
650  IF KEY$ < > "L" THEN 670
660 SYMBOL = 3:X = X - 10: GOTO 7
60
670  IF KEY$ < > "P" THEN 690
680 FLAG = 1: GOSUB 1000: GOTO 53
0
690  IF KEY$ = "Q" THEN 1090
700  REM  NEXT STATEMENT PROTECTS
 FROM KEYING ERROR
710  IF KEY$ < > "E" THEN 570
720  HCOLOR= 0:FLAG = 0: GOSUB 10
00
730  REM  SET UP PRE-PLOT STATUS
740 KSVE$ = KI$: HCOLOR= 3: GOTO
500
750  REM  ADJUST 3-BIT VECTOR FOR
 PLOT
760  IF KSVE$ = "P" THEN SYMBOL =
SYMBOL + 4
770  REM  LOAD 3-BIT VECTOR INTO
BYTE
780 CYCLE = CYCLE + 1
790  IF CYCLE < > 1 THEN 810
800 BYTE = SYMBOL: GOTO 490
810  IF CYCLE < > 2 THEN 900
820 BYTE = BYTE + 8 * SYMBOL
830  REM  PROTECT AGAINST PREMATU
RE END-OF-RECORD
840  IF BYTE > 7 THEN 490
850  REM  ENTER DUMMY RIGHT MOVE
AND STORE BYTE.
860 BYTE = BYTE + 8: POKE ADDR,BY
TE:ADDR = ADDR + 1
870  REM  ENTER UP MOVE AND DUMMY
 LEFT MOVE IN NEW BYTE
880 BYTE = 24:CYCLE = 2: GOTO 490

890  REM   IF THIRD 3-BIT VECTOR
IS A MOVE ONLY,FINISH BYTE;E
LSE LOAD BYTE INTO TABLE AND
 STORE 3-BIT VECTOR IN NEXT
BYTE.
```

```
900  IF SYMBOL > 3 THEN 930
910 BYTE = BYTE + 64 * SYMBOL
920  REM  STORE BYTE
930  POKE ADDR,BYTE:ADDR = ADDR +
1
940  REM  STORE 3-BIT VECTOR IN N
EXT BYTE IF NEEDED
950  IF SYMBOL = 0 OR SYMBOL > 3 THEN
980
960  REM  PREPARE FOR NEXT BYTE G
ET NEXT 3-BIT VECTOR
970 CYCLE = 0: GOTO 490
980 CYCLE = 1:BYTE = SYMBOL: GOTO
490
990  REM  PLOT ROUTINE
1000  FOR Y2 = Y - 3 TO Y + 3 STEP
6: HPLOT X - 1,Y2 TO X + 1,Y
2: NEXT
1010  FOR Y2 = Y - 2 TO Y + 2 STEP
4: HPLOT X - 2,Y2 TO X + 2,Y
2: NEXT
1020  FOR Y2 = Y - 1 TO Y + 1: HPLOT
X - 3,Y2 TO X + 3,Y2: NEXT
1030  REM   TURN OFF CURSOR IN PL
OTTED SQ.
1040  IF X = XS AND Y = YS THEN  RETURN

1050  XDRAW 1 AT X,Y: RETURN
1060  REM  PREPARE BYTE FOR QUIT
1070  REM  CLOSE OUT BYTE FOR MOV
E-ONLY
1080  IF KSVE$ < > "P" THEN 1150

1090  REM  USE PLOT-THEN-UP VECTO
R TO END
1100  IF CYCLE < > 2 THEN 1120
1110  POKE ADDR,BYTE:ADDR = ADDR +
1
1120  IF CYCLE < > 1 THEN 1140
1130 BYTE = BYTE + 32: GOTO 1150
1140 BYTE = 4
1150  POKE ADDR,BYTE:ADDR = ADDR +
1
1160  REM  ADD RECORD MARK,DISPLA
Y NEW SHAPE
1170  POKE ADDR,0:ADDR = ADDR + 1
: XDRAW N AT 200,75
1180  INPUT "SAVE SHAPE? Y/N ";KI
$
1190  IF KI$ = "Y" THEN 1220
1200 N = N - 1: GOTO 190
1210  REM  GET INDEX FOR NEXT FRE
E BYTE
1220 N = N + 1:ADDR = ADDR - ASVE
1230  IF N < MAX THEN 1270
1240  PRINT "WARNING: TABLE FULL
WITH THIS SHAPE "
1250  IF N > MAX THEN 1310
1260  REM  STORE INDEX IN DIRECTO
RY
1270  POKE ASVE + 2 * N,ADDR - 25
6 *  INT (ADDR / 256)
1280  POKE ASVE + 2 * N + 1, INT
(ADDR / 256)
1290  INPUT "DONE? Y/N ";KI$
1300  IF KI$ = "N" THEN 160
1310  INPUT "SAVE TABLE? Y/N ";KI
$
1320  REM  RESPONSE PROTECTED AGA
INST RANDOM KEY HIT
1330  IF KI$ = "Y" THEN 1360
1340  IF KI$ = "N" THEN 1370
1350  GOTO 1310
1360  PRINT D$;"BSAVE" + NAME$ +
",A" +  STR$ (ASVE) + ",L" +
 STR$ (ADDR)
1370  END
```

8: The Display Program

```
10  REM  ***DISPLAY SHAPE TABLE*
***
20  REM  LOAD NUMERALS SHAPE FILE

30  PRINT : PRINT : PRINT "HIT AN
Y KEY FOR EACH PAGE OF TABLE
"
40 D$ =  CHR$ (4): PRINT D$;"NOMO
N C,I,O"
50  PRINT D$;"BLOAD NUMERALS,A200
00,D2"
60  REM  SET UP POINTER TO NUMERA
LS
70 NHI = 78:NL = 32
80  REM  GET TABLE FOR DISPLAY
90  INPUT "SHAPE TABLE NAME ";NAM
E$
100  INPUT "STARTING ADDRESS ";AD
DR
110  REM  SET UP POINTER TO SHAPE
 TABLE
120 AHI =  INT (ADDR / 256):ALO =
ADDR - 256 * AHI
130  REM  GET NO. OF SHAPES FOR D
ISPLAY
```

```
140 NN =  PEEK (ADDR)
150  REM  INITIALIZE SCREEN
160  HGR : POKE  - 16302,0
170  HCOLOR= 3: SCALE= 1: ROT= 0
180  FOR I = 1 TO NN
190 IMOD = I - 36 *  INT (I / 36)

200  IF IMOD <  > 1 THEN 350
210  GET KEY$
220  REM  SCLEAR SCREEN AND CREAT
E GRID
230  REM   GRID WILL HOLD 36 SHAP
ES
240  CALL 62450
250  HPLOT 0,0 TO 269,0 TO 269,19
0 TO 0,190 TO 0,0
260  FOR L = 45 TO 269 STEP 45
270  FOR J = 0 TO 190 STEP 10
280  HPLOT L,J
290  NEXT J: NEXT L
300  FOR L = 30 TO 190 STEP 30
310  FOR J = 0 TO 269 STEP 10
320  HPLOT J,L
330  NEXT J: NEXT L
340  REM  CALCULATE GRID SQUARE C
OORDS
350  IF IMOD = 0 THEN IMOD = 36
360 ROW =  INT ((IMOD - 1) / 6)
370 COL = IMOD - 6 * ROW - 1
380 C1 =  INT (I / 100)
390 C2 = I - 100 * C1
400 C2 =  INT (C2 / 10)
410 C3 = I - 10 *  INT (I / 10)
420  POKE 232,NLO: POKE 233,NHI
430 C1 = C1 + 2:C2 = C2 + 2:C3 =
C3 + 2
440  IF C1 = 2 THEN 460
450  DRAW C1 AT 45 * COL + 5,30 *
ROW + 7
460  IF C2 = 2 AND C1 = 2 THEN 49
0
470  DRAW C2 AT 45 * COL + 10,30 *
ROW + 7
480  DRAW C3 AT 45 * COL + 15,30 *
ROW + 7
490  REM  NOW GET SHAPES
500  POKE 232,ALO: POKE 233,AHI
510  DRAW I AT 45 * COL + 30,30 *
ROW + 15
520  NEXT I
530  GET KEY$
540  TEXT
550  END
```

MICRO Product Review

As outlined briefly in MICRO 18:58, MICRO is instituting a review policy/procedure in which a "Review Staff" of volunteer, independent, qualified computerists will be asked to review 6502 based products:
hardware, software, books, etc.

Product Submission

While any 6502 based product is "fair game" for a review, we plan to handle products whose manufacturer requests a review first. The procedure is simple. Fill out the attached "Review Request Form" and send it to us. We will select products for review from the submitted forms and select a reviewer from our Review Staff. We will contact the reviewer to make sure he is willing to review the product, has time to do the review, has no "conflict of interest", etc. Once a reviewer is set, we will contact you to supply a sample of the product you wish reviewed. This will be sent to the reviewer. Upon receipt of the review, we will send a copy to you. You will have a chance to make comments about the review, clear up any misunderstandings, point out items that may have been overlooked, discuss significant changes and improvements being planned, etc. If valid errors in the review are pointed out, we will get back to the reviewer and see that all points are clearly covered and understood before the review is printed. This does **not** mean that you will have any editorial rights in regards to what is finally printed. It does mean that you will have opportunities to help insure that the review adequately covers the important features of your product and that minor problems will not be blown out of perspective.

In the event that a review is, in our opinion, biased — too bad or too favorable — we may have a second reviewer evaluate the product. Our goal is to be able to present to the MICRO readers a review that is as complete and unbiased as possible. We think that this will be an important service both to the readers and to the manufacturers. Here will be a way to get a fair evaluation about your product out to thousands of interested readers (customers?). Since the review will be by an independent reviewer, the material will have a lot more impact than a "self-serving" product news release.

Supplying Samples

The manufacturer must furnish a sample of the material to be reviewed. In the case of books and software, the sample will normally be kept by the reviewer and not returned. In the case of hardware, the sample will be returned. We would like to suggest, in the case of hardware, that the reviewer be permitted to purchase the hardware at dealer price or some other reasonable discount if he so desires. If such a discount is acceptable to you, this should be mentioned on the Review Request Form. The reviewer will not be informed about any product discount until after the review is finalized in order to prevent any unwanted bias from entering into the review.

All of this effort to provide a new method for getting reviews is intended to produce accurate, unbiased, believeable reviews. These will be of service to the readers and manufacturers. We hope that you will submit your 6502 based product for review. It should benefit everyone.

Robert M. Tripp

MICRO Review Request Form

Manufacturer: ....

Address: ....

City: .... State: .... Zip: ....

Phone: Days: .... Evenings: ....

Name of Product: ....

Description of Product: ....

....

....

6502 Systems it works with: ....

....

Special Equipment/Software Required: ....

....

Retail Price: .... First Delivery: .... Number Delivered: ....

I declare that I am the manufacturer, or legal representative of the manufacturer, and request that MICRO produce a review on this product. I agree to provide a sample of the product for review. I understand that software or book samples will not be returned, but that hardware samples will normally be returned. I am/am not willing to provide a discount on the hardware sample to the reviewer.

Signed: .... Date: ....

Title: ....

Please complete, using additional pages if necessary, and return to: **MICRO, P.O. Box 6502, Chelmsford, MA 01824**

Relocating PET BASIC Programs

Michael Tulloch, Ph.D.
103 White Circle
Niceville, FL 32578

Some important details are presented about the organization of PET BASIC and a technique is provided to permit BASIC programs to be shifted to different memory locations.

Have you ever wanted to time share with your PET? How about ROM routines in BASIC? You can do both of these and more by writing "shifted" BASIC programs and redirecting PET's monitor. First, I'm going to very briefly describe where PET stores BASIC programs and where the important pointers are located. Then, I'll tell you how to ENTER and RUN BASICprograms antwhere in PET's lower 32K of memory. Finally, I'll give you a practical example.

Initialization

When PET's monitor initializes memory, either with power on or by executing SYS(64824), a bunch of things happen. PET writes decimal 36 (24 HEX or screen symbol $) into each memory location. After each location is written the same location is read. PET thus actively determines its contigous memory size by finding the first non-36 location. Since the lower page (decimal 0 to 1032) is used as a scratch pad, PET starts its memory check at decimal 1024. Memory size is stored in 134, 135, as two bytes. The first byte is low and the second byte is high, standard 6502 format. After determining memory size, PET initializes its BASIC program memory to ready it for a BASIC program. Table 1 gives these values. Just why these location hold what they do requires a detailed description of how PET BASIC works. Such a description is too long for this article. But, this peculiar pattern is necessary.

Scratch Pad Usage

The scratch pad memory also has some other important values. As I mentioned above, memory size was stored in 134, 135. Now six additional values are inserted. These values are called pointers. They point to locations in the program memory where the monitor goes during BASIC execution and/or program entry. These pointers are BASIC start address, simple variables star address, array variables start address, available space start address, top of strings and bottom of strings. Let's see just where these pointers are stored and what their initial values are. The BASIC pointer, which is stored in memory location 122, 123, is initialized to 1025. This pointer tells the monitor where to start storing and reading BASIC program statements. The simple variables pointer, which is stored in memory location 124, 125, is initialized to 1028. This pointer tells the monitor where the simple variables start. The array variables pointer, which is stored in memory locations 126, 127, is also initialized to 1028. This pointer is always equal to the simple variables pointer until an array variable is DIMensioned. It performs a similiar function to that of the simple variables pointer. The available space pointer, stored in memory locations 128, 129, is initialized to 1028. Top and bottom of string variable pointers are stored in memory locations 132, 133, and 130, 131 respectively. Strings are stored top down while both simple and array variables are stored bottom up. Figure 1 shows how PET's monitor arranges the BASIC program and variables in memory. To store a BASIC program in a different place in memory we have to change the values of these pointers. Let's assume for a moment that these seven pointers have been changes. This will force the monitor to try to store a program, entered from the keyboard, in a location defined by pointer values. However, there is one more thing which must be done. The area which has been defined by the seven pointers must be initialized as shown in table 1. Once that has been done everything is ready. The program is entered in the normal fashion. When completed, the program can be executed without any further adjustments. It can be RUN or reLOADed as long as PET isn't turned off. Programs entered this way aren't in the normal place for a BASIC program.

Saving Shifted Programs

Saving a shifted program isn't as straightfoward as you might wish. For those lucky enough to have Version 2 ROMs it's easy. All you have to do is call the machine language monitor and SAVE the program like you would SAVE a machine language program. The rest of us have to resort to tricking the PET.

```
S S

8192 TOP OF     +----------+    255*PEEK(135)+PEEK(134)
     MEMORY     + S        +    256*PEEK(133)+PEEK(132)
                +  T       +
     TOP OF     +   R      +
     STRINGS    +    I     +
                +     N    +
     BOTTOM     +      GS  +
     STRINGS    +----------+    256*PEEK(131)+PEEK(130)
                +    $     +
                +    $     +
1028 AVAILABLE  +----------+    255*PEEK(129)+PEEK(128)
     SPACE      +    ^     +
     START      +    ^     +
                +   A(3)   +
1028  ARRAY     +----------+    255*PEEK(127)+PEEK(126)
     VARIABLES  +    ^     +
                +    D     +
                +    C     +
                +   A2     +
                +          +
1028  SIMPLE    +----------+    255*PEEK(125)+PEEK(124)
     VARIABLES  +    ^     +
                +    ^     +
                +  BASIC   +
                + PROGRAM  +
1025  BASIC     +          +
      START     +----------+    255*PEEK(123)*PEEK(122)
>S
```

Figure 1: **Pet Memory Map and Pointer Locations**

When SAVE is used from the keyboard the routine initializes one of the cassette buffer pointers to 1024. POKEing the starting address of the shifted program doesn't work (and finding this out delayed this article several months-I was SAVING all of memory from 1024 up)! Fortunately there is a way around this problem. IN "Commodore PET Users Club Newsletter", Vol. 1, Issue 4&5 there is a program which demonstrates just what we need to trick the PET. Table 2 lists the required lines. By using SYS to access the SAVE routine we can bypass the initialization. The listed code can be used either as direct commands or as part of a program.

How it Works

Line 1 sets the first address for cassette #1. Lines 2 and 3 set the high(B) and low (A) bytes of the start address. Lines 4 and 5 set, in a similiar fashion, set the end address to the value of the simple variables start address. This address is the same as the end of the BASIC program. Line 6 calls the SAVE routine. There is one disadvantage-this simple approach leaves the program name undefined. "$$$" or " " is assigned as the file name. Shifted programs can be LOADed, and VERIFIED just like regular BASIC programs. However, if the monitor has reinitialized memory, any attempt to LIST or RUN a shifted program will fail. If a shifted program has been SAVEd, PET turned off and back on, and the shifted program is reLOADed it still cannot be LISTed or RUN.

How come? I did just say it would RUN when entered from the keyboard. Well, it's those seven pointers. When PET SAVEs a program, any program, it stores an image of the program as it appears in RAM. However, not all of the pointer values are stored on the tape. Since PET uses a compiled (not really compiled like FORTRAN but actually compacted) listing, it must also store the forward chain addresses along with the compacted code. Each BASIC statement has a forward chain address. This forward chain address points to the forward chain address of the next BASIC statement. Therefore, the program must be stored in exactly the same memory location from which it originally came. Forward cahin addressing is absolute rather than relative. If PET has reinitialized its pointers, the BASIC pointer is pointing to the normal BASIC location. Upon loading a BASIC program tape under keyboard control the SV, AV, AS registers are loaded with data from the tape. Unfortunately, the monitor assumes BASIC programs will always start at 1025. Therefore when PET is asked to RUN or LIST, the monitor will start looking at 1025. It won't find a program. To use a shifted program after it has been LOADed back into the PET the BASIC pointer must be changed.

There are several ways to do this. One can simply POKE the correct values into the pointer memory locations. This works, but if you make a mistake the PET will "go away" when you try to RUN the program. With version I ROMs the only thing you can do is turn the PET off. There may be a good side to this approach; it can be used as a neat way to protect a program. Without some clever PEEKing at RAM and without understanding how to set the pointers based upon that PEEKing, the program won't run. Another approach is to have a machine language program do the required initialization. With this approach several shifted programs can be RUN at once. To call a specific program you can use the USER (X) or SYS commands. The machine language program does the rest. I'll give an example of a simple routine like this in the last section.

Memory Location		Value	
Base 10	Hex	Base 10	Hex
1024	400	0	0
1025	401	0	0
1026	402	0	0
1027	403	36	24
1028	404	73	49
1029	405	0	0
1030	406	139	8B
1031	407	0	0
1032	408	0	0
1033	409	0	0
1034	40A	0	0
1035	40B	0	0
1036	40C	36	24

Table 1: **Pet BASIC Initialization Values**

```
 100 POKE241,1:REMDEVICE #(1=TAPE 1)
 105 A=PEEK(122):B=PEEK(124):
     REM BASIC START POINTER
 110 POKE247,A:POKE248,B:
     REM SAVE FROM POINTER
 120 B=PEEK(124):POKE229,B:REM BASIC
 130 B=PEEK(125):POKE230,B:REM END
 140 SYS63153:
     REM ROM SAVE ROUTINE
READY.
```

Figure 2

Shifted programming has several advantages but there are also some pitfalls. I'm sure that I haven't found them all. I'll tell you about those that I've fallen into, and Murphy will find some new ones for you. As a first example, let's take the case where shifted programs are loaded in under keyboard control. When this is done, all memory above 1024 is reinitialized. Any shifted programs already in memory are 36'd out. The only way to prevent this is to adjust the top of memory pointer so that it points below the existing shifted programs. This must be done before atempting to LOAD from the keyboard. Shifted (or normal) programs LOADed under program control do not 36 out memory. But the first part of memory may be set up to receive BASIC. In addition, pointers aren't changed.

Another pitfall is the tendency for PET to "go away". Any error in pointer setup will usually cause this problem. It is the rule rather than the exception. Version 2 ROMs are rumored to allow a warm reset. Unfortunately, they aren't available for the old 8K PETs yet.

A third pitfall is really just the result of careless programming. The available space within any program should be reduced as much as possible. Program space includes variable and string space. Although my PET has 16K of memory (half in BETSI), I've found it easy to over-run memory or to overlap programs. If multiple BASIC programs are to coexist, a memory map and some planning are necessary. I don;t have a dynamic adjustment routine. Perhaps someone familiar with the PET montior could adapt its program adjustment software. It works on normal programs and it sure is fast. PET uses the routine whenever new lines are added or old lines deleted. If variable pointers are the same for all programs and if assignment statements are used to initialize all programs, then several programs might be able to share the variable working area. I haven't tried a lot of this, but it does work in simple cases. This technique will allow FORTRAN like passed variable subroutines, support BLOCK type statements and conserve a lot of memory.

So much for the pitfalls, here's some of the good news. The shifted program technique can be used for BASIC programs to coexist with Commodore's tape machine language monitor. Sure, you'll be able to buy a new set of ROMs that have the monitor—someday. But you can have nearly the same thing now. You may need an additional routine to transfer the bottom of page one (0A-22 hex) memory back and forth between machine language monitor and BASIC usage. Both BASIC and the machine language monitor want this part of memory for scratch pad.

What else can be done with shifted BASIC programs? ROM BASIC programs, truly modular development, library routines, and lots more. Now that BASIC programs can be placed wherever you want them, your imagination is the only limit.

apple computer

Apple II Reference Manual	$10.00
Apple Soft Manual	10.00
Programmer's Guide (Computer Station)	5.95
Apple II Monitor Peeled	9.95
Software Directory for Apple	
• Business, Finance & Utility	4.95
• Games, Demo, Utility	4.95
Best of Contact '78	2.50
Programming in PASCAL (Grogono)	9.90

INTERFACE CARDS

Prototyping/Hobby Card	$ 24 00
Parallel Printer Interface Card	180.00
Communications Card & D825 Connector Cable	225.00
High-Speed Serial Interface Card	195.00
Language System with Pascal (48K RAM & Disk II Required)	495.00
Applesoft II Firmware Card	200.00
16 Input Analog Card	295 00

ACCESSORIES

Disk II—Drive Only	495.00
Disk II—Drive & Controller (32K Min. RAM Recommended)	595 00
Vinyl Carrying Case	30.00
Tape Recorder	40.00
Programmers Aid No. 1 Firmware (For Use with Integer BASIC)	50.00
Clock/Calendar Card	199 00
Auto—Start ROM Package (For Apple II Only)	65.00
Digikitizer Pad by Talos (Kitform)	499.00

High Resolution Light Pen	199.00
Micromodem (D.C. Hayes)	379 00
12" B/W Leebex Monitor	149.00
Cable from Monitor to Apple II	9.95
13" Color TV Compatible with Apple II	290.00
Sup-R-Modulator (RF)	25.00

SOFTWARE FOR APPLE II

PASCAL from Programma	49.95
FORTH	49.95
LISP—from Apple Software 8k No. 3	N/C
LISA—Interactive disk assembler	34 95
WHATSIT—Excellent conversational data base manager	32K 100 00 48K 125.00
SARGON-Champ of 2nd West Coast Computer Faire	19.95
APPLE PIE—Excellent text editor	24.95
FORTE—Music editor in hires	19.95
FASTGAMMON—Excellent backgammon game with graphics	Tape 20.00 Disk 25.00
APPLE 21—Excellent blackjack game	9.95
BRIDGE CHALLENGER—Computer bridge	14.95

FINANCIAL MANAGEMENT SYSTEM
- Accounts Payable
- Accounts Receivable
- Inventory Control
- $200 Each Package
- Ledger Processing
- Payroll
- $800 Complete
- $10 for Manual

PRINTER SPECIALS FOR APPLE AND PET

TRENDCOM 100 with Interface for Apple or PET	450.00
LITE PEN used with TV or monitor screen	34.95
ALF Music Synthesizer Boards	265.00
Supertalker	279.00
Anadex DP-8000 with tracter 8" paper width and interface to Apple	1050.00
Centronics 779-2 for Apple II with parallel interface	1245 00

SOFTWARE (Send for complete Software Catalog $1.00)

Dow Jones Portfolio Evaluator/Stock Quote Reporter Disk	50.00
Microchess 2.0 Chess Disk	25 00
Disk Utility Pack with DOS 3.2	25 00
The Controller (General Business System)	625.00
Apple Post (Mailing List System)	49 95
Bowling Program Diskette	15 00
The Cashier (Retail Store Management	250 00
Checkbook Cassette	20.00
Applesoft II Language & Demo Cassette	20.00
RAM Test Tape with Manual	7.50
Finance 1-2 Cassette Package	25 00
Datamover/Telepong Cassette (Com. Card & Modem Req'd)	7 50
Microchess 2.0 Chess Tape	20 00
Bowling Program Tape	15 00
Pascal with Language System (48K RAM & Disk II Required)	495 00

New for Apple Computer Owners at Low ComputerWorld Prices

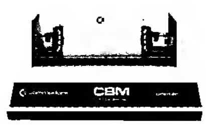
8" Disk Drives with housing **$1295.00** for single drive **$1895.00** for dual drive

apple computer

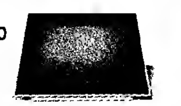
Joystick for Apple II **Only $39.95 each.**
ComputerWorld's Complete Library of 600 Apple II programs **Only $60.00**

A PROFESSIONAL BUSINESS SYSTEM

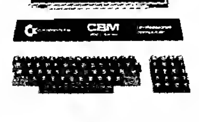
commodore CBM™



CMB 2022
$995



CBM 2040
$1295



2001-16B
$995

2001-8	$795
2001-16N	$995
2001-32N	$1295
2001-32B	$1295
External Cassette	$95
PET to IEEE Cable	$39.95
CBM 2023 Printer	$849
IEEE to IEEE Cable	$49.95

PET™

Commodore PET Service Kit	$30.00
Beeper—Tells when tape is Loaded	24.95
Petunia—Play music with PET	29.95
Video Buffer—Attach another display	29.95
Combo—Petunia and Video Buffer	49.95

SOFTWARE FOR PET

Mirrors and Lenses	19.95
The States	14.95
Real Estate 1 & 2	59.95
Momentum and Energy	19.95
Projectile Motion	19.95
Mortgage	14.95
Dow Jones	7.95
Petunia Player Sftwr	14.95
Checkers and Baccarat	7.95
Chess	19.95
Series and Parallel Circuit Analysis	19.95
Home Accounting	9.95
BASIC Math	29.95
Game Playing with BASIC Vol. I, II, III	9.95 each

Join Now

Become a member of ComputerWorld's RAYGAMCO Computer Discount Club.
By being a RAYGAMCO Member you receive substantial discounts on every item you purchase, including all hardware, software, accessories, even books and paper! You will also receive a monthly newsletter with all the latest available for your particular computer system, and much, much more — exclusive to RAYGAMCO Members only!

Here's how to join.
Nothing to buy. Simply complete the self-addressed postcard in this magazine with name, address, and your computer system. You'll receive your personalized RAYGAMCO Computer Discount Club Membership Card in the mail with information on how to use your card when ordering for big savings!

Charter RAYGAMCO Members' Special. Join now and receive 20% OFF of any purchase.*

*20% offer expires December 24, 1979. Offer is valid for RAYGAMCO Members only.

ComputerWorld

A RAYGAM COMPANY

6791 Westminster Ave., Westminster, CA 92683 (714) 891-2587

If You Treat It Nicely It Won't Byte

Jack Robert Swindell
P.O. Box 8193
Canton, OH 44711

Tools and techniques for using the Superboard II are presented — including a Double Disassembler. This program gives a lot of information about each byte of memory, not just the opcode. Several other Superboard features are discussed.

I selected the Superboard II for use as an intelligent terminal in a PDP-II system. It enables the designer of a distributed processing system to take a number of liberties due to the speed and power of each distributed branch. Before this multi-processor system can come into full operation, a number of things need to be discovered about the internal workings of the Superboard. This article describes some of the tools, techniques and discoveries found on the road to the goal. I hope you find them as useful as I have.

In order to really gain an understanding of the inner workings, a disassembler or something similar will be required, as the monitor leaves a lot to be desired. The listing in figure 1 uses about 3.6K of memory, i.e. you need at least 5K to run it. It is a combination mnemonic lister and intelligent disassembler. The leftmost column will *always* print a mnemonic, thusly treating each and every instruction as though it were only one byte in length. The rightmost column attempts to decipher whether the instruction is one, two, or three bytes in length and differentiate its print to distinguish op-codes from their operands. Columns two and three are the address and op-code in decimal form to help when using PEEK and POKE at later times. The fourth column prints any valid ASCII characters that it finds to help with the recognition of text or buried cues when the disassembler "gets confused" and has to re-sync itself or might need some help.

The reason I mention manual re-sync is that one soon grows weary of seeing "resync??????" time and time again when the program is running through a giant table of either string data or numeric data. Of course it *will* re-sync...but why waste the paper? On to columns five and six; these have the address and op-code in hexidecimal format to help when looking in books (which are nearly all in hex now). The rightmost and seventh column is what it is all about.

The seventh column is the intelligent column. It attempts to convey to you its interpretation of what it's reading out of memory. It does not resequence the order of bytes for printing when looking at a multi-byte instruction as many disassemblers do. I didn't deem it necessary at the time. To illustrate my point, look at illustration 2. The JSR at hex 0222 has AB directly following it and CD two bytes later. A little human

translation saves much software. Illustration 2 is a nonsense program, there only to show you what it looks like when it runs and how it runs. Hex lines 0228 to 022C show what happens when the program runs into something it doesn't recognize; the string prompt "CARP?". The response is always the same: it prints the first line it didn't recognize followed by the row of question marks and then four more lines without trying to assign an "intelligent" op-code or do anything else except get ready to re-sync (or try) on the fifth byte after the initial unlock. If this byte also lacks a valid mnemonic the process is repeated until it finally drops out and finds one.

After you start the program, it will ask you for the addresses of the lowest byte and the highest byte that you want it to try to disassemble. This must be input in decimal form as the program has no provisions for a hexidecimal to decimal converter. The next thing that we'll do is examine the program to help you to see how it works and where the various routines are. Lines 100 through 730 comprise the data table. Each data statement holds the information to decode four different instructions of 6502 op-codes and also "fillers" to tell the program when a non-existant instruction is found. The format is "MNEMONIC", NUMBER OF BYTES for that instruction. If it is a non-existant instruction then the data statement for it will read: "?",9. Since as far as I know there aren't any nine byte 6502 instructions, it sticks out quite well amidst a forest of ones, twos, and threes.

Now it's time for the fun part. Line 1020 inputs the address range to be worked on. Lines 1040 and 1050 print the header. Line 1070 sets the major loop which cycles through the op-codes one byte at a time. 1100 to 1120 cause the data table to be scanned until the correct op-code is found. The second statement in line 1120 tells the program the total number of lines to print without mnemonics when it gets out of sync. 1130 to 1150 print the leftmost four columns. 1220 to 1260 control the program's intelligence and tell it when and when not to try and print a mnemonic in the rightmost column.

A GOSUB 1500 with 0 to 15 in H will return the hexadecimal equivalent in H$. GOSUB 1400 with 0 to 255 in D returns the hex equivalent in I$. GOSUB 1300 with 0 to 65535 in R returns 0000 to FFFF (hex) in J$. These last three routines are "quick and dirty" but may be of some use to you at a later time. The data table is easily modified to allow for future expansion. Standard Rockwell/Sybex mnemonics are used except for the use of hyphens as opposed to commas (the data statements wouldn't like these too well I fear).

```
Input low&high addresses of block to be listed:Decimal? 546,565


MNE           A-DEC    O-DEC    ASCII A-HEX    O-HEX  MNE(if valid)


JSR           546      32             0222     20     JSR
?             547      171            0223     AB     *** AB ***
CMP           548      205            0224     CD     *** CD ***

JSR           549      32             0225     20     JSR
?             550      18             0226     12     *** 12 ***
?             551      52       4     0227     34     *** 34 ***

?             552      67       C     0228     43     ?
Resync??????????????????????????????????????????????????????????????
EOR-I-X       553      65       A     0229     41     *** 41 ***
?             554      82       R     022A     52     *** 52 ***
BVC           555      80       P     022B     50     *** 50 ***
?             556      63       ?     022C     3F     *** 3F ***

BRK           557      0              022D     00     BRK

PHA           558      72       H     022E     48     PHA

TXA           559      138            022F     8A     TXA

CMP-IMM       560      201            0230     C9     CMP-IMM
?             561      67       C     0231     43     *** 43 ***

BNE           562      208            0232     D0     BNE
SBC-O-P-X     563      245            0233     F5     *** F5 ***

NOP           564      234            0234     EA     NOP

NOP           565      234            0235     EA     NOP
```

Figure 2

```
 50000 FORD=BTOB+11*CSTEPC:POKED,32:NEXTD:A$=STR$(A):E=LEN(A$)
 50010 FORF=BTOB+(E-1)*CSTEPC:POKEF,ASC(MID$(A$,(F-B+C)/C,1)):NEXTF
 50020 RETURN
OK
```

Figure 3

Numeric To Video Conversion

This short BASIC routine will enable you to print numeric variables on your video monitor while your software is busy generating real-time graphics. See figure (3). The opeeration is not overly complex. First the program clears the screen positions which are going to have new characters placed there. This is done by POKEing blanks there with a FOR-NEXT loop. The number that you are going to display is first converted to a string with the STR$ function. The length of the resultant string is found with the LEN function. MID$ is used with a FOR-NEXT loop to dissect the string into individual characters which are converted to the correct values to be POK'd into the screen memory with the ASC function.

The display is a fixed format which uses the 12 screen positions: the mantissa sign, 6 digits of mantissa with a decimal point, exponent sign and two digits of exponent. Or ±0.00000E±00.

```
100 FORD=53240TO54271:POKED,32:NEXTD
110 B=53776
120 A=RND(2)*10^(RND(4)*10)
130 C=1:GOSUB50000
140 FORC=34TO30STEP-1:GOSUB50000:NEXTC
150 C=-1:GOSUB50000
160 FORC=-34TO-30:GOSUB50000:NEXTC
170 GOTO120

OK
```

Figure 4

Beware of blank characters when examining strings for video conversion! 12 screen positions ARE required! It is important to remember that when the number is pushed into the display the starting video address will always be the mantissa sign position. This can be any screen address but beware of overlapping when you try and print off the edge of the screen. The number to be displayed need not always be displayed in a left to right fashion. By changing the video incrementing factor many print angles become possible. Here is a listing in a clock fashion with the mantissa sign at the starting video address.

1 o'clock	−31	7 o'clock	31
2 o'clock	−30	8 o'clock	30
3 o'clock	1	9 o'clock	−1
4 o'clock	34	10 o'clock	−34
5 o'clock	33	11 o'clock	−33
6 o'clock	32	12 o'clock	−32

To run this routine place the number which you wish merged to the display in register A. Load the starting video address in register B. Put the video incrementing factor in register C. Gosub 50000. Once A, B, and C are loaded they remain intact after program execution.

A picture is worth a thousand words (2K bytes?). Load and run the program in figure (4) to see both how all the different display angles look and what happens when a scientific notation display is caused to overlap the edge of the display when run at a steep angle. Make sure you load figure (3) or it will try and call a non-existant subroutine.

On-Screen Expose'

Did you know that there is a graphics/control character that you can print on the screen by just pressing two keys? There is! Control G will create the character that you see when you try to type in a line that's a bit too long. You can type it into a string just like it was a letter or symbol. As an added bonus, if you have a printer tied in, it will ring its bell...instant prompt.

I have one more thing of interest for you before I return to bury myself in my favorite world of semiconductors and software. The location (in page zero) of the on screen text begins at 19 decimal and continues up to 90 decimal which always contains a zero when examined. Therefore 71 bytes can be defined, the 72nd is a zero. To see what I mean do the following in command mode:

1) Press Return (to make sure everything is terminated).

2) Hold down the space bar until the screen starts to show the control G characters mentioned earlier.

3)Press Return (this clears the on screen text internally).

4) Type perfectly:FORS = 19T090:?CHR$(PEEK(S));:NEXTS.

5) Press Return.

Now do you see what I mean? Happy computing, that's all for now. Would anyone want to hear about a Superboard speedup? Almost 2MHZ or double speed and it doesn't alter the I/O baud rates, however, none of the OSI RAM chips could cut the mustard. If you want an article on this, write! Bye.

Fig. (Listing) 1.

```
10 REM Double Disassembler
20 REM Written by
30 REM Jack Robert Swindell
40 REM August 23, 1979
100 DATA"BRK",1,"ORA-I-X",2,"?",9,"?",9
110 DATA"?",9,"ORA-O-P",2,"ASL-O-P",2,"?",9
120 DATA"PHP",1,"ORA-IMM",2,"ASL-A",1,"?",9
130 DATA"?",9,"ORA",3,"ASL",3,"?",9
140 DATA"BPL",2,"ORA-I-Y",2,"?",9,"?",9
150 DATA"?",9,"ORA-O-P-X",2,"ASL-O-P-X",2,"?",9
160 DATA"CLC",1,"ORA-Y",3,"?",9,"?",9
170 DATA"?",9,"ORA-X",3,"ASL-X",3,"?",9
180 DATA"JSR",3,"AND-I-X",2,"?",9,"?",9
190 DATA"BIT-O-P",2,"AND-O-P",2,"ROL-O-P",2,"?",9
200 DATA"PLP",1,"AND-IMM",2,"ROL-A",1,"?",9
210 DATA"BIT",3,"AND",3,"ROL",3,"?",9
220 DATA"BMI",2,"AND-I-Y",2,"?",9,"?",9
230 DATA"?",9,"AND-O-P-X",2,"ROL-O-P-X",2,"?",9
240 DATA"SEC",1,"AND-Y",3,"?",9,"?",9
250 DATA"?",9,"AND-X",3,"ROL-X",3,"?",9
260 DATA"RTI",1,"EOR-I-X",2,"?",9,"?",9
270 DATA"?",9,"EOR-O-P",2,"LSR-O-P",2,"?",9
280 DATA"PHA",1,"EOR-IMM",2,"LSR-A",1,"?",9
290 DATA"JMP",3,"EOR",3,"LSR",3,"?",9
300 DATA"BVC",2,"EOR-I-Y",2,"?",9,"?",9
310 DATA"?",9,"EOR-O-P-X",2,"LSR-O-P-X",2,"?",9
320 DATA"CLI",1,"EOR-Y",3,"?",9,"?",9
330 DATA"?",9,"EOR-X",3,"LSR-X",3,"?",9
340 DATA"RTS",1,"ADC-I-X",2,"?",9,"?",9
350 DATA"?",9,"ADC-O-P",2,"ROR-O-P",2,"?",9
360 DATA"PLA",1,"ADC-IMM",2,"ROR-A",1,"?",9
370 DATA"JMP-I",3,"ADC",3,"ROR",3,"?",9
380 DATA"BVS",2,"ADC-I-Y",2,"?",9,"?",9
390 DATA"?",9,"ADC-O-P-X",2,"ROR-O-P-X",2,"?",9
400 DATA"SEI",1,"ADC-Y",3,"?",9,"?",9
410 DATA"?",9,"ADC-X",3,"?",9,"?",9
420 DATA"?",9,"STA-I-X",2,"?",9,"?",9
430 DATA"STY-O-P",2,"STA-O-P",2,"STX-O-P",2,"?",9
440 DATA"DEY",1,"?",9,"TXA",1,"?",9
450 DATA"STY",3,"STA",3,"STX",3,"?",9
460 DATA"BCC",2,"STA-I-Y",2,"?",9,"?",9
470 DATA"STY-O-P-X",2,"STA-O-P-X",2,"STX-O-P-X",2,"?",9
480 DATA"TYA",1,"STA-Y",3,"TXS",1,"?",9
490 DATA"?",9,"STA-X",3,"?",9,"?",9
500 DATA"LDY-IMM",2,"LDA-I-X",2,"LDX-IMM",2,"?",9
510 DATA"LDY-O-P",2,"LDA-O-P",2,"LDX-O-P",2,"?",9
520 DATA"TAY",1,"LDA-IMM",2,"TAX",1,"?",9
530 DATA"LDY",3,"LDA",3,"LDX",3,"?",9
540 DATA"BCS",2,"LDA-I-Y",2,"?",9,"?",9
550 DATA"LDY-O-P-X",2,"LDA-O-P-X",2,"LDX-O-P-Y",2,"?",9
560 DATA"CLV",1,"LDA-Y",3,"TSX",1,"?",9
570 DATA"LDY-X",3,"LDA-X",3,"LDX-Y",3,"?",9
580 DATA"CPY-IMM",2,"CMP-I-X",2,"?",9,"?",9
590 DATA"CPY-O-P",2,"CMP-O-P",2,"DEC-O-P",2,"?",9
600 DATA"INY",1,"CMP-IMM",2,"DEX",1,"?",9
610 DATA"CPY",3,"CMP",3,"DEC",3,"?",9
620 DATA"BNE",2,"CMP-I-Y",2,"?",9,"?",9
630 DATA"?",9,"CMP-O-P-X",2,"DEC-O-P-X",2,"?",9
640 DATA"CLD",1,"CMP-Y",3,"?",9,"?",9
650 DATA"?",9,"CMP-X",3,"DEC-X",3,"?",9
660 DATA"CPX-IMM",2,"SBC-I-X",2,"?",9,"?",9
670 DATA"CPX-O-P",2,"SBC-O-P",2,"INC-O-P",2,"?",9
680 DATA"INX",1,"SBC-IMM",2,"NOP",1,"?",9
690 DATA"CPX",3,"SBC",3,"INC",3,"?",9
700 DATA"BEQ",2,"SBC-I-Y",2,"?",9,"?",9
710 DATA"?",9,"SBC-O-P-X",2,"INC-O-P-X",2,"?",9
720 DATA"SED",1,"SBC-Y",3,"?",9,"?",9
730 DATA"?",9,"SBC-X",3,"INC-X",3,"?",9
800 REM End of data table
900 CLEAR
1000 PRINT"6502 Double Disassembler - 1979 - J. Swindell"
1010 PRINT
1020 INPUT"Input low&high addresses of block to be listed:Decimal";P,Q
1030 PRINT:PRINT:PRINT:PRINT
1040 PRINT"MNE";TAB(15);"A-DEC";TAB(25);"O-DEC";TAB(33);"ASCII";
1050 PRINTTAB(39);"A-HEX";TAB(48);"O-HEX";TAB(55);"MNE(if valid)"
1060 PRINT:PRINT
1070 FORU=PTOQ
1080 M=PEEK(U)
1090 RESTORE
1100 FORO=OTOM
```

```
1110 READM$,N
1120 NEXTO:IFN=9THENN=5
1130 PRINTM$;TAB(15);U;TAB(25);M;TAB(33);
1140 IFM<32ORM>126THENPRINTCHR$(32);
1150 IFM>=32ANDM<127THENPRINTCHR$(M);
1160 R=U
1170 GOSUB1300
1180 PRINTTAB(39);J$;TAB(48);
1190 D=M
1200 GOSUB1400
1210 PRINTI$;TAB(55);
1220 IFV=0THENT=N
1230 IFV=0THENPRINTM$
1240 IFV=0ANDT=5THENPRINT"Resync";:FORB=1TO58:
     PRINT"?";:NEXTB:PRINT"?"
1250 IFV>0THENPRINT"*** ";I$;" ***"
1260 V=V+1:IFV=T THENV=0:PRINT
1270 NEXTU:PRINT:PRINT:PRINT:PRINT
1280 PRINT"END OF RUN":PRINT:PRINT
1290 END
1300 D=INT(R/256)
1310 GOSUB1400
1320 J$=I$
1330 D=R-D*256
1340 GOSUB1400
1350 J$=J$+I$
1360 RETURN
1400 E=INT(D/16)
1410 F=D-E*16
1420 H=E
1430 GOSUB1500
1440 I$=H$
1450 H=F
1460 GOSUB1500
1470 I$=I$+H$
1480 RETURN
1500 IFH<10THENH$=MID$(STR$(H),2,1)
1510 IFH<0THENH$="0"
1520 IFH=10THENH$="A"
1530 IFH=11THENH$="B"
1540 IFH=12THENH$="C"
1550 IFH=13THENH$="D"
1560 IFH=14THENH$="E"
1570 IFH>=15THENH$="F"
1580 RETURN
```

Sharpen Your AIM

Robert E. Babcock
1706 Fawcett Ave.
White Oak, PA 15131

A collection of four programs are presented which enhance the capabilities of the basic AIM 65. These programs improve hex loading, clear memory, move memory and slow down the display.

Recently several Rockwell AIM-65 microcomputer systems were purchased for use in teaching courses in microprocessors and microcomputers at the campus of the Pennsylvania State University at which I teach. These were intended to supplement the KIM-1 systems which have been used for that purpose for the past three years.The press of other activities has prevented more than intermittent exposure to the full capabilities of the AIM-65; however, some basic impressions and evaluations are possible.

Overall, the impression has been highly favorable. First, due to the similarity with the KIM-1, the AIM has been easy to learn. Even students with virtually no exposure to any type of microcomputer have had little difficulty in learning to use the system effectively. In this regard, the documentation provided with the AIM-65 is excellent. The AIM-65 Microcomputer User's Guide is easy to follow and has a sizeable number of examples to clarify concepts stated in the material related to a portion of the system or its operation. Identification of many of the most useful subroutines and their characteristics has proved to be a special blessing. The clock program used as an application example at the end of the manual involves virtually every mode of operation. It provides an excellent base for understanding the system and in addition serves as a firm foundation for a flexible data sampling and logging system. Although a few errors exist in the User's Manual, most are of minor consequence.

Second, the extensive monitor program has a great many features not generally found in a system of this price class. These features make it possible to program the AIM more rapidly and with fewer errors than is possible for an essentially identical program using the KIM-1.The features which come to mind most readily are the mnemonic entry capability, the disassembler, and the text editor. The printer with its hard copy put the topping on the physical attributes of the system. Less visible, but equally as convenient, are the cassette interface with its much higher speed and flexibility when compared with the KIM-1. The ability to use the KIM format permits the application of many KIM programs to the AIM. Finally, the 20 character display with the ability to use alphanumerics expands the capabilities of the AIM-65.

No system is completely without its shortcomings and the AIM is no exception. Fortunately, the shortcomings are few and most are easily corrected. One of the problems arises from the fact that in the memory modify mode,(/), the program is returned to the system monitor after four entries. While all that is necessary to return to the modify mode is to again press (/), often when entering a program from a hex dump format or entering hex values into a table or entering a short ASCII message statement, it is easy to forget to re-enter (/). The short program shown below, HEX LOAD, uses the same format as the M followed by (/) process but automatically remains in the modify mode until terminated by an ESC. There is a printout of the entered characters and the address of the lowest byte just as in the normal operation. The only difference is that it is no longer necessary to enter (/) after each four entries. To use HEX LOAD, begin execution at 0600 (or the beginning address selected if in a different location) by the usual entries, "(*)=0600", RETURN,"G", RETURN. The display will show "= ". Enter the address at which hex entries are to start, RETURN, and the starting address will be displayed with the prompt "^". Make the desired hex entries as a continuous string, then terminate with ESC.

```
HEX LOAD
(K)*=0600
/20

0600 20 JSR EAAE
0603 20 JSR E83E
0606 A0 LDY #00
0608 20 JSR EA5D
060B 90 BCC 0613
060D C9 CMP #20
060F D0 BNE 0623
0611 F0 BEQ 061B
0613 20 JSR EB78
0616 F0 BEQ 061B
0618 4C JMP EB33
061B 20 JSR E83E
061E C8 INY
061F C0 CPY #04
0621 D0 BNE 0608
0623 20 JSR E2CD
0626 20 JSR EA13
0629 20 JSR E2DB
062C 20 JSR E83E
062F D0 BNE 0606
```

```
SLOW DIS
(K)*=0200
/38

0200 A9 LDA #4B
0202 20 JSR E97A
0205 A9 LDA #2A
0207 20 JSR E97A
020A 20 JSR EAAE
020D B0 BCS 0200
020F 20 JSR E5D7
0212 20 JSR EB37
0215 20 JSR E785
0218 20 JSR EA24
021B 20 JSR F46C
021E AD LDA A425
0221 38 SEC
0222 65 ADC EA
0224 8D STA A425
0227 90 BCC 022C
0229 EE INC A426
022C 20 JSR EA24
022F 20 JSR E907
0232 20 JSR E790
0235 F0 BEQ 023D
0237 20 JSR 0240
023A 4C JMP 021B
023D 4C JMP E1A1
0240 A9 LDA #10
0242 85 STA AC
0244 A9 LDA #00
0246 8D STA A00B
0249 A9 LDA #FF
024B 8D STA A008
024E 8D STA A009
0251 A9 LDA #20
0253 2C BIT A00D
0256 F0 BEQ 0253
0258 AD LDA A008
025B C6 DEC AC
025D D0 BNE 0249
025F 60 RTS
```

ZERO PAGE LOCATIONS USED:

00AC	Timing Loops
00EA	Length (Used by monitor ROM)

The second difficulty is an annoyance with the speed at which disassembly occurs when the printer is not in operation. This mode of operation is sometimes desirable to conserve paper while debugging or while checking for a particular part of a program. The program left, SLOW DIS, introduces about a 1 second delay between steps during disassembly without the printer. Location 0241 can be modified to change the speed as desired. Execute the program in the normal way using (*)=0200, RETURN, "G", RETURN. The display will indicate "K* = ". Enter the starting address of the material to be disassembled and the number of steps as in normal operation. If an indefinite number of steps was selected by "SPACE", then the program must be terminated by ESC.

One of the major advantages of the AIM-65 over the KIM-1 and other similar systems using 7-segment read-out displays (limited to six digits), is the relative ease of using meaningfully prompted programs which eliminate the need to record or remember the proper addresses into which data must be entered to initiate the program. With prompting, the required information can be asked for, inserted, and stored in appropriate locations under program control. Two utility programs, CLEAR and MOVER, included below, are of the prompted type. MOVER is a data transfer program capable of moving any amount of data either forward or backward to a designated starting address. Execution of the progam results in a prompting message of "OLD FROM=" to elicit the entry of the starting address of the data to be moved. After the address has been entered and RETURN activated, "TO=" calls for the ending address of the data to be moved. When RETURN is again used, theprompt "NEW FROM=" appears to bring about entry of the starting address at which the moved data is to start. This time RETURN causes execution of the move process, completion of which is indicated by a cleared display except for the normal " " at the left side of the display. Similarly, CLEAR uses prompting messages, "CLR FROM=" and "TO=" to obtain the limiting addresses of the area into which zeros or any other designated character may be entered. The area can be of any size.

A general breakdown of the features of these two programs can be used to show the various sections and their functions. In CLEAR, the program from 0300 through 0314 provides the prompt message generation; 0315 through 0330 contains the address input and storage functions; 0331 through 033D contains the calculation of the high and low order bytes of the length of the area involved; and the remainder of the program performs the actual data storage procedure. Location 0340 may be modified to any value with which it is desired to load a selected memory area. Locations 035F - 0361 contain the "CLR" message.

```
CLEAR

(K)*=0300
/46

0300 20 JSR EA13
0303 A0 LDY #00
0305 B9 LDA 035F
0308 48 PHA
0309 29 AND #7F
030B 20 JSR E97A
030E C8 INY
030F 68 PLA
0310 10 BPL 0305
0312 20 JSR E83E
0315 20 JSR E7A3
0318 AD LDA A41C
031B 85 STA 00
031D AD LDA A41D
0320 85 STA 01
0322 20 JSR E7A7
0325 B0 BCS 0322
0327 AD LDA A41C
032A 85 STA 02
032C AD LDA A41D
032F 85 STA 03
0331 38 SEC
0332 A5 LDA 02
0334 E5 SBC 00
0336 85 STA 04
0338 A5 LDA 03
033A E5 SBC 01
033C F0 BEQ 034C
033E AA TAX
033F A9 LDA #00
0341 A8 TAY
0342 91 STA (00),Y
0344 C8 INY
0345 D0 BNE 0342
0347 E6 INC 01
0349 CA DEX
034A D0 BNE 0342
034C E6 INC 04
034E A9 LDA #00
0350 A0 LDY #00
0352 91 STA (00),Y
0354 C8 INY
0355 C4 CPY 04
0357 D0 BNE 0352
0359 20 JSR EA13
035C 4C JMP E1A1

(M)=035F 43 4C D2
```

MOVER

```
(K)*=0200
/96

0200 20 JSR EA13          0270 E6 INC AA
0203 A0 LDY #00           0272 A6 LDX A9
0205 20 JSR 02B8          0274 F0 BEQ 0286
0208 20 JSR E7A3          0276 B1 LDA (A2),Y
020B 20 JSR F910          0278 91 STA (AA),Y
020E 20 JSR E7A7          027A 88 DEY
0211 B0 BCS 0208          027B C0 CPY #FF
0213 20 JSR EA13          027D D0 BNE 0276
0216 AD LDA A41A          027F C6 DEC A3
0219 85 STA A0            0281 C6 DEC AB
021B AD LDA A41B          0283 CA DEX
021E 85 STA A1            0284 D0 BNE 0276
0220 AD LDA A41C          0286 E6 INC A8
0223 85 STA A2            0288 B1 LDA (A2),Y
0225 AD LDA A41D          028A 91 STA (AA),Y
0228 85 STA A3            028C 88 DEY
022A A0 LDY #04           028D C6 DEC A8
022C 20 JSR 02B8          028F D0 BNE 0288
022F 20 JSR E83E          0291 20 JSR EA13
0232 20 JSR E7A3          0294 4C JMP E1A1
0235 AD LDA A41C          0297 A0 LDY #00
0238 85 STA A4            0299 A6 LDX A9
023A AD LDA A41D          029B F0 BEQ 02AB
023D 85 STA A5            029D B1 LDA (A0),Y
023F 38 SEC               029F 91 STA (A4),Y
0240 A5 LDA A2            02A1 C8 INY
0242 E5 SBC A0            02A2 D0 BNE 029D
0244 85 STA A8            02A4 E6 INC A1
0246 A5 LDA A3            02A6 E6 INC A5
0248 E5 SBC A1            02A8 CA DEX
024A 85 STA A9            02A9 D0 BNE 029D
024C 18 CLC               02AB E6 INC A8
024D A5 LDA A4            02AD B1 LDA (A0),Y
024F 65 ADC A8            02AF 91 STA (A4),Y
0251 85 STA AA            02B1 C8 INY
0253 A5 LDA A5            02B2 C4 CPY A8
0255 65 ADC A9            02B4 D0 BNE 02AD
0257 85 STA AB            02B6 F0 BEQ 0291
0259 38 SEC               02B8 B9 LDA 02C6,Y
025A A5 LDA A4            02BB 48 PHA
025C E5 SBC A0            02BC 29 AND #7F
025E 85 STA A6            02BE 20 JSR E97A
0260 A5 LDA A5            02C1 C8 INY
0262 E5 SBC A1            02C2 68 PLA
0264 85 STA A7            02C3 10 BPL 02B8
0266 90 BCC 0297          02C5 60 RTS
0268 A0 LDY #FF
026A C6 DEC A3            (M)=02C6 4F 4C 44 A0
026C C6 DEC AB            (.) 02CA 4E 45 D7
026E E6 INC A2
```

ZERO PAGE LOCATIONS USED:

CLEAR

0000	Start ADDR Low
0001	Start ADDR High
0002	Ending ADDR Low
0003	Ending ADDR High
0004	Length Low

MOVER

00A0	OLD Start ADDR Low
00A1	OLD Start ADDR High
00A2	OLD Ending ADDR Low
00A3	OLD Ending ADDR High
00A4	NEW Start ADDR Low
00A5	NEW Start ADDR High
00A6	Move Distance Low
00A7	Move Distance High
00A8	PGM Length Low
00A9	PGM Length High
00AA	NEW Ending ADDR Low
00AB	NEW Ending ADDR High

A similar examination of MOVER will show that the segment from 0200 through 023E generates the prompting messages by way of a subroutine at 02B8 - 02C5, obtains the requested addresses and stores them. From 023F through 0266 is found the calculation procedures for the length of the data to be moved, determination of the new ending address, and decision as to whether movement is forward or backward. Movement upward in address by starting at the end and working back to the start is contained in 0268 through 0294, while movement downward in address is handled from 0297 through 02B7. The "OLD" and "NEW" messages are contained in 02C6 - 02CC.

These programs have been found very useful in assisting an already powerful system to be even more responsive to the desires of the programmer. Other programs which would be very helpful would be the ability to insert an instruction into the middle of a program with automatic movement of the remainder to make room, as is done in the text editor and some assemblers. Related would also be a deletion procedure with automatic closure. Not enough time has been available to accomplish these programs. Perhaps later...

Receipt of the 8K basic ROM's for the AIM-65 has finally occurred after a lengthy wait. Not enough opportunity has arisen to delve into that aspect of the AIM very deeply, as yet. A brief exposure has made a very favorable impression. The addition of the BASIC makes the AIM-65 into exactly what its name implies; a self-contained Advanced Interactive Microcomputer.

An Additional I/O Interface for the PET

Interfacing a VIA 6522 to your PET is simple.

Kevin Erler
P.O. Box 3032
Edson, Alberta TOE OPO
Canada

The 6522 VIA chip has a lot of interesting features, however, many of them are on the "PB"side of the chip. The Commodore PET does not have the "PB" lines on its user port, only the "PA" lines. The following interface gives not only the wanted "PB" lines but also an extra set of "PA" lines &CB1, CB2, CA1, & CA2.

The Hardware

The circuit itself uses only a 6522 VIA and two 7411's. It is mostly direct interfacing, other than the address lines which had to be decoded. Once built, it connects directly to the Memory Expansion Port.

The interface (in figure 1) is designed to occupy the top 16 bytes of RAM. It should be noted here that adding another interface is as simple as changing the address decode. For example, by placing an inverter on "BA4" (see figure 2)the circuit would occupy the 16 bytes of RAM just under the top 16 bytes. (note-if you build both of the circuits from figures 1 & 2 you would have two VIA's and would be using the top 32 bytes of RAM). The original circuit is shown in figure 1.

The Software

After connecting it, operation is very simple. The addresses concerned and what they are follows. (for the circuit shown in figure 1)

32752 - QRB
32753 - QRA
32754 - DDRB
32755 - DDRA
32756 - TIL-L TIC-L
32757 - TIC - H
32758 - TIL-L
32759 - TIL-H
32760 - T2L-L T2C-L
32761 - T2C-H
32762 - SR
32763 - ACR
32764 - PCR
32765 - IFR
32766 - IER
32767 - ORA (no handshake)

The operation is as with other VIA--- PEEK POKE etc., only with the previously listed addresses.

Note--for the addresses which operate the circuit in figure 2, simply subtract 16 from each address.

Output Example

To create a tone on CB2 for the circuit in figure 1;

POKE 32763, 16 (ACR)
POKE 32762, 15 (SR)
POKE 32760, 155 (Timer 2) for the circuit in figure 2.
POKE 32747, 16 (ACR)
POKE 32746, 15 (SR)
POKE 32744, 155 (Timer 2)

For further specs. on the "PB" port of the 6522, refer to the 6522 data sheet.

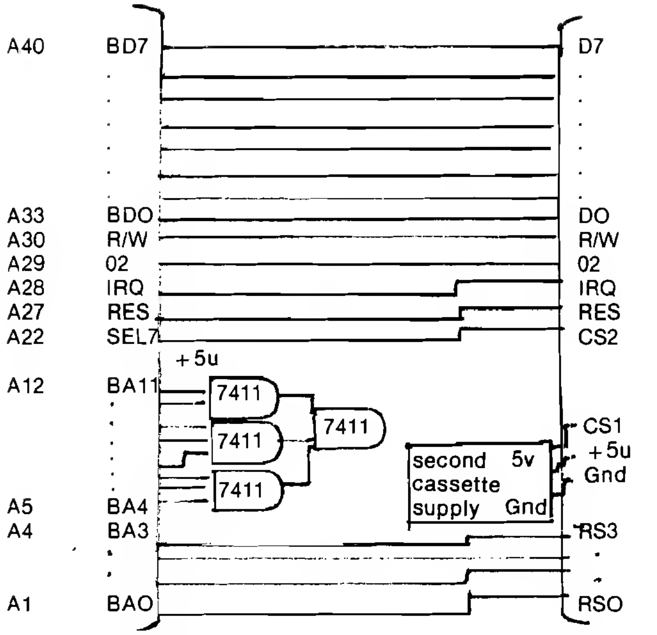


Figure 1: **Interface designed to occupy top 16 bytes of RAM**

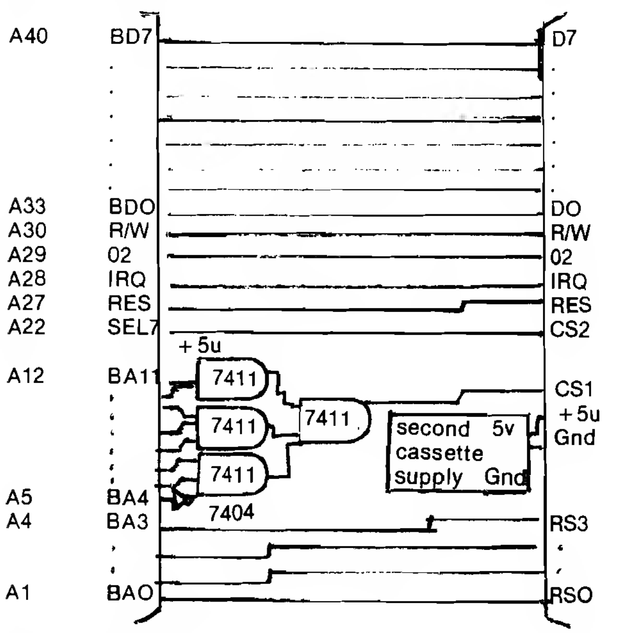


Fig. 2: **Interface designed to occupy 16 bytes just under top 16 bytes of RAM.**

A 60 × 80 Life for the PET

Werner Kolbe
Hardstr. 77
CH 5432 Neuenhof
Switzerland

Have you ever wished that your PET display was bigger, especially when playing the Game of LIFE? Here is a method of providing a moveable window that permits you to examine any portion of an area that is: 'Larger than Life'.

When you have played some time with the 25 x 40 LIFE by Dr. Covitz, you will find that the area is too small for many patterns to expand. Therefore I decided to write a program which gives them more space. As I still wanted to use the nice round CHRS(81) dots as cell symbols, I decided to show only a section of the whole area on the screen. The screen is practically used as a movable window which can be shifted in 8 directions by the number keys 1 to 9. The '5' is used to bring it back into the center.

Program Description

The BASIC part of the program does the following: Line 0 sets the memory pointers to prevent BASIC from destroying the machine code and to restore the "end of BASIC" pointer in dec 124, 125. Then in sub. 100, a short explanation, is given. The cells are set on the screen in the input mode with A$, where A$ is not used. Line 4 to 10 do the shifting of the screen versus the Life area. The pointer PA which determines the displayed section is changed by the pokes into 2940, 2941. Line 3 again raises the memory pointer and lifts the "end of BASIC" pointer over the end of the machine code. Thus it is possible to save the whole program including machine code by a simple SAVE.

The machine program starts at the location hex 0A80. The memory used as Life-area starts at 0C51 and ends at 1F11. All necessary pointers are located in the BASIC input buffer from 0029 to 003F. They are initialized with the subroutine INIT from TBL2 starting at OB6A. The pointer P9 points (indexed by Y) to the place which is currently investigated. The pointers P1 to P8 point to the neighboring places. PA points to the upper left corner of the displayed section and PS to the start of the screen. CNT is a page counter.

Cells are represented by bit 7 of the memory. The cells for the next generation are stored in bit 6. Subroutine CLEAR sets everything to zero. Then in NE the screen is inspected and if a 51 is found, bit 7 is set in the associated memory place. Subroutine INPDEX increases the pointers PS by dec 40 and PA by dec 80 if one row has gone through (Y running). By storing hex 34 respectively hex 3C into E811 the screen is switched off resp. on again to avoid "snow". After START the new generation is computed. The number of neighbors is counted by inspection of the neighboring places and decreasing X if bit 7 is set. If the life condition is found for the next generation, bit 6 is set in the memory place. When one page is worked through, all high values of the pointers P1 to P9 are incremented. The pages are counted by CNT. With RESTORE, the old generation is pushed out by a left shift, and the new one

Listing 1

```
0 POKE135,10:POKE124,216:POKE125,006:CLR:GOSUB100
1 SYS2730:GETA$:IFA$=""THEN1
2 IFA$=" "THENINPUTA$:SYS2691:GOTO1
3 IFA$="E"THENPOKE135,32:POKE124,131:POKE125,11:END
4 IFA$="5"THENOX=Ø:OY=Ø
5 A=VAL(A$):OX=OX+OX(A):OY=OY+OY(A)
6 IFOX>20THENOX=20
7 IFOX<-20THENOX=-20
8 IFOY>18THENOY=18
9 IFOY<-18THENOY=-18
10 P=4533+OX+OY*80:PH=INT(P/256):PL=P-PH*256:POKE2940,PL:POKE2941,PH
11 POKE515,255:GOTO1
100 PRINT"chcdcdcd          *** LIFE 60X80 *** cdcdcdcdcdcd
101 POKE2940,181:POKE2941,17
102 FORA=0TO9:READOX(A),OY(A):NEXT
104 PRINT"cdcdcdPUT THE CELLS WITH '●' ON THE SCREEN.
106 PRINT"cdSTART WITH 'RET.', STOP WITH 'SPACE'.
107 PRINT"cdEND WITH 'E'.
108 PRINT"cdMOVE THE WINDOW WITH 1 TO 9
            cdTHE 5 CENTERS IT.
109 PRINT"cdcdcdrvsPRESS ANY KEY.
110 GETA$:IFA$=""GOTO110
111 PRINT"chcdcdcdcdcdcdcdcdcd":INPUTA$:SYS2688:RETURN
120 DATA0,0,2,-2,0,-2,-2,-2,2,0,0,0,-2,0,2,2,0,2,-2,2
cd = Cursor down     ch = Clear-Home     rvs = Reverse
```

comes from bit 6 into bit 7. Since there does not exist an indirect addressing for the ASL command, I had to use the absolute indexed to increment the argument directly. Finally, TSCR throws the cells on the screen with 51's if bit 7 is set and 20's (blanks) else. The RTS returns the control back to BASIC. For one generation the programs needs about 1/2 second. The speed may be slowed down by a waiting loop in BASIC.

Combining BASIC and Machine Code

If you have entered the machine code, type in NEW (but don't switch off) and enter the BASIC code. If you have finished, find out the actual values of the "end of BASIC" pointer in dec 124 and 125 by PEEK commands. If they differ from 216 resp.6, the pokes in line 0 must be changed. Before a run, this POKE must contain the actual value of the pointer, after the last change in the BASIC program.

To save everything on tape enter: POKE 124, 130: POKE 125, 11: CLR and then SAVE "LIFE 60*80". With the POKE, the "end of BASIC" pointer is raised beyond the end of the machine code and thus with the save, both program parts are combined. When running the program, line 0 restores the old values of the pointer. The program can be loaded and run like any other program. Only if changes are made in BASIC, line 0 must be updated.

Listing 2

```
0A80   20 57 0B   FS      JSR CLEAR
0A83   20 4C 0B   NE      JSR INIT
0A86   A9 34              LDA=34
0A88   8D 11 E8           STA E811
0A8B   A2 18              LDX=18
0A8D   A0 27      LP1     LDY=27
0A8F   B1 3D      LP2     LDA (PS),Y
0A91   C9 51              CMP=51
0A93   D0 04              BNE 04
0A95   A9 80              LDA=80
0A97   D0 02              BNE 02
0A99   A9 00              LDA=00
0A9B   91 3B              STA (PA),Y
0A9D   88                 DEY
0A9E   10 EF              BPL LP2
0AA0   20 34 0B           JSR INPDEX
0AA3   10 E8              BPL LP1
0AA5   A9 3C              LDA=3C
0AA7   8D 11 E8           STA E811
0AAA   78         START   SEI
0AAB   20 4C 0B           JSR INIT
0AAE   A2 01      LP3     LDX=1
0AB0   B1 29              LDA (P1),Y
0AB2   10 01              BPL 01
0AB4   CA                 DEX
0AB5   B1 2B              LDA (P2),Y
0AB7   10 01              BPL 01
0AB9   CA                 DEX
0ABA   B1 2D              LDA (P3),Y
0ABC   10 01              BPL 01
0ABE   CA                 DEX
0ABF   B1 2F              LDA (P4),Y
0AC1   10 01              BPL 01
0AC3   CA                 DEX
0AC4   B1 31              LDA (P5),Y
0AC6   10 01              BPL 01
0AC8   CA                 DEX
0AC9   B1 33              LDA (P6),Y
0ACB   10 01              BPL 01
0ACD   CA                 DEX
0ACE   B1 35              LDA (P7),Y
0AD0   10 01              BPL 01
0AD2   CA                 DEX
0AD3   B1 37              LDA (P8),Y
0AD5   10 01              BPL 01
0AD7   CA                 DEX
0AD8   8A                 TXA
0AD9   10 10              BPL TOD
0ADB   C9 FE              CMP=FE
0ADD   F0 06              BEQ LBN
0ADF   30 0A              BMI TOD
0AE1   B1 39              LDA (P9),Y
0AE3   10 06              BPL TOD
0AE5   A9 40      LBN     LDA=40
0AE7   11 39              ORA (P9),Y
0AE9   91 39              STA (P9),Y
0AEB   88         TOD     DEY
0AEC   D0 C0              BNE LP3
0AEE   A2 12      INPTS   LDX=12
0AF0   F6 28      LP4     INC TBL-1,X
0AF2   CA                 DEX
0AF3   CA                 DEX
0AF4   D0 FA              BNE LP4
0AF6   C6 3F              DEC CNT
0AF8   10 B4              BPL LP3
0AFA   A9 12      RESTR   LDA=12
0AFC   85 3F              STA CNT
```

```
0AFE   A9 0C              LDA=P9H
0B00   8D 05 0B           STA P9H'
0B03   1E 51 0C    LB4    ASL P9',X
0B06   CA                 DEX
0B07   D0 FA              BNE LB4
0B09   EE 05 0B           INC P9H'
0B0C   C6 3F              DEC CNT
0B0E   10 F3              BPL LB4

0B10   A9 34       TSCR   LDA=34
0B12   8D 11 E8           STA E811
0B15   A2 18              LDX=18
0B17   A0 27       LB5    LDY=27
0B19   B1 3B       LB6    LDA (PA),Y
0B1B   10 04              BPL 04
0B1D   A9 51              LDA=51
0B1F   D0 02              BNE 02
0B21   A9 20              LDA=20
0B23   91 3D              STA (PS),Y
0B25   88                 DEY
0B26   10 F1              BPL LB6
0B28   20 34 0B           JSR INPDEX
0B2B   10 EA              BPL LB5
0B2D   A9 3C              LDA=3C
0B2F   8D 11 E8           STA E811
0B32   58                 CLI
0B33   60                 RTS     BACK TO BASIC

0B34   18          INPDEX CLC
0B35   A9 50              LDA=50
0B37   65 3B              ADC PAL
0B39   85 3B              STA PAL
0B3B   90 03              BCC 03
0B3D   E6 3C              INC PAH
0B3F   18                 CLC
0B40   A9 28              LDA=28
0B42   65 3D              ADC PSL
0B44   85 3D              STA PSL
0B46   90 02              BCC 02
0B48   E6 3E              INC PSH
0B4A   CA                 DEX
0B4B   60                 RTS
```

```
0B4C   A0 17       INIT   LDY=17
0B4E   BE 69 0B    LB7    LDX TBL2-1,Y
0B51   96 28              STX TBL-1,Y
0B53   88                 DEY
0B54   D0 F8              BNE LB7
0B56   60                 RTS

0B57   20 4C 0B    CLEAR  JSR INIT
0B5A   E6 3F              INC CNT
0B5C   A9 00              LDA=00
0B5E   91 29       LB8    STA (P1),Y
0B60   88                 DEY
0B61   D0 FB              BNE LB8
0B63   E6 2A              INC P1H
0B65   C6 3F              DEC CNT
0B67   10 F5              BPL LB8
0B69   60                 RTS

TBL2                      TBL
0B6A   00          TBL2   0029   00   P1L
0B6B   0C                 002A   0C   P1H
0B6C   01                 002B   01   P2L
0B6D   0C                 002C   0C   P2H
0B6E   02                 002D   02   P3L
0B6F   0C                 002E   0C   P3H
0B70   50                 002F   50   P4L
0B71   0C                 0030   0C   P4H
0B72   52                 0031   52   P5L
0B73   0C                 0032   0C   P5H
0B74   A0                 0033   A0   P6L
0B75   0C                 0034   0C   P6H
0B76   A1                 0035   A1   P7L
0B77   0C                 0036   0C   P7H
0B78   A2                 0037   A2   P8L
0B79   0C                 0038   0C   P8H
0B7A   51                 0039   51   P9L
0B7B   0C                 003A   0C   P9H
0B7C   B5                 003B   B5   PAL
0B7D   11                 003C   11   PAH
0B7E   00                 003D   00   PSL
0B7F   80                 003E   80   PSH
0B80   12                 003F   12   CNT
```

EASYWRITER,* the 1st true Word Processor for the Apple!*

Are you looking for the best Word Processor for your Apple? Well we are so sure you'll choose **EasyWriter** that we've prepared this ad to help you make your decision **EASY**. Check out these powerful features:

- Incremental spacing to support your Qume, Diablo, or Spinwriter
- Character oriented (No line numbers to deal with)
- Menu selectable routines for all known printers and interfaces
- Word wrap around on screen for continual text entry
- Our own new high speed DOS (Twice as fast as Apple's)
- Of course full editing, disk, and printer commands
- Subscripting, Superscripting, and MORE MORE MORE . . .

The straight facts make EasyWriter the only logical choice. By the same people who brought you WHATSIT. Available at your local computer store or our new California office!

	EasyWriter*	Dr. Memory*	Big Edit*	Apple Pie*	Super-Text*		
Incremental Spacing	X						
Character Oriented Screen	X				X		
Printer Interface Routines	X						
Word Wrap Around	X				X		
Editing Commands	X	X	X	X	X		
High Speed Disk	X						
50 Pages of Text per Disk	X						
Subscripting & Superscripting	X				X		
Your Choice							

793 Vincente
Berkeley, CA 94707
(415) 525-4046

146 N. Broad Street
Griffith, IN 46319
(219) 924-3522

It Isn't Software Until it Works!

A perfect Christmas gift!

Information Unlimited Software™ Inc.

ORDER FORM

CUT ALONG DOTTED LINE

Name:

Company:

Address:

City: **State:** **Zip:**

I would like more information on:

ITEM	QUANTITY	PRICE	TOTAL
Easy Writer (EZ 2)		$ 99.95	
Whatsit Model A-1 (Apple)		125.00	
Whatsit Model CP-2 (CP/M)		150.00	
Whatsit Model NS-3 (N Star)		100.00	
Whatsit Manual		25.00	
EasyWriter Manual		30.00	
Subtotal			$
CA Residents add 6% sales tax			$
Shipping & Handling			$ 2.50
GRAND TOTAL			$

Master Charge or Visa Number:

***EasyWriter**
EasyWriter is a TM of Cap'n Software
***Dr. Memory**
Dr. Memory is a TM of Muse
***Big Edit**
Big Edit is a TM of Gravey, Martin & Sampson, Inc.
***Apple Pie**
Apple Pie is a TM of Programma International, Inc.
***Super-Text**
Super-Text is a TM of Muse
Apple
Apple is a TM of Apple Computers, Inc.
Whatsit
Whatsit is a TM of Computer Headware

Applesoft Program Relocation

George S. Guild, Jr.
117 Cardinal Drive
Hampton, VA 23664

Here is a simple technique to change the program storage space when using Applesoft.

Integer BASIC has commands to set boundaries for both the program upper limit (HIMEM) and data lower limit (LOMEM). This gives Integer BASIC users total freedom to protect areas of memory for HIRES graphics and/or machine language subroutines. Applesoft however, uses *fixed* program storage, and uses HIMEM and LOMEM only to set the upper and lower boundaries of stored data. This lack of flexibility can result in problems when using Applesoft.

For example, RAM Applesoft users were forever limited to 4K of program space, when they wanted to use HIRES graphics, even if 48K of memory was available. Setting LOMEM to $6000 (24576) preserves all 4K for programming with data saved above the HIRES page 2. Users of the Heuristics Speechlab have found that the firmware stores its data starting at $800 (2048). This data would overwrite any BASIC program created by the ROM Applesoft, limiting its use to Integer BASIC.

The sequence of commands shown in the insert allows Applesoft users to overcome this limitation. First decide where you want your program to start, i.e. the lowest address of the program. For example, if you want to use the memory space above HIRES page 2, this address would be $6000 (24576) for the start of program storage. Store $00 to the first three bytes here and then set the program pointer ($67, 68) to the starting address plus one.

Programs loaded will now start at $6000 until you reset the pointer or reload/reinvoke Applesoft. CLEAR, NEW, LOAD, and RESET do not affect this pointer. Change the start address and program pointer for your requirements.

Do not set the program pointer lower than $801 for ROM Applesoft or $3001 for RAM Applesoft because doing so will either interfere with the text screen area ($400 to $800) or overwrite the RAM interpreter which is stored at $800 to $2FFF.

Users of DOS versions earlier than DOS 3.2 may have to execute a CALL 3314, for disk Applesoft, or a CALL 54514, for ROM Applesoft, in order to update programs loaded from disk. DOS 3.2 does the required CALL automatically. Cassette systems have no such problem.

]SAVE	If the program you wish to relocate is in memory you must save it first.
]"Reset"	Enter monitor.
*6000:00 00 00	Store zeroes at beginning of new program space. If omitted, strange syntax errors occur.
*67:01 60	Set program pointer to new start address plus one. Note that pointer is stored in low byte first, then high byte, as usual for 6502 microprocessor.
*3D0G	Disk system return to BASIC. (Cassette system/ROM Applesoft: Control-B; RAM Applesoft: 0G)
]NEW	Initialize Applesoft
]LOAD	Program will be loaded starting at address $6000.

KIM and SYM Format Cassette Tapes on APPLE II

Steven M. Welch
309 S. Sunset
Longmont, CO 80501

Now you can swap programs and data between your APPLE and any AIM, SYM or KIM via cassette I/O.

Many KIM and SYM owners have graduated to bigger and better 6502 systems as their needs and financial situations changed. If you are one of these people, and find that your KIM is sitting in the corner gathering dust because your APPLE is so much easier to work with, read on. With this program, you can use your APPLE as a "host computer" for assembly language program development and then "down load" the finished program into your single board computer (SBC). Just like the big boys! Not only will you make better use of your several hundred dollar investment, but you will also have the bonus of a new set of computer jargon to bore your friends. The value of developing assembly language programs in this fashion cannot be fully appreciated until you use the APPLE to develop a sizeable program for the SYM or KIM. The many miseries of hand assembling magically disappear. The constant verbal self-abuse which generally accompanies calculator keyboard entry and debugging quickly becomes a fading memory. Have you ever forgotten to initialize a loop counter only to realize it 300 bytes of hand assembly later?

The program listed here was produced to fill a need; a need to develop a large program on a SYM. I estimate that we have saved an absolute minimum of 2 man-months in the development of a 1500 byte program by using the APPLE for entry, debugging and assembling. Also, having a real assembler easily available to us, we have written better code and have not needed the numerous patches and kludges which inevitably crop up when one writes large programs in machine code. At the University of Colorado at Boulder, where I am employed, we are developing a microprocessor-controlled Charge Coupled Photo Diode [CCPD] spectrographic detector for the Sommers-Bausch Observatory using a SYM-1 computer. Although this is a very nice SBC, it lacks certain features which are highly desireable in a computer that will be used for program development, e.g., fast mass storage, an assembler, text editor, ASCII keyboard, and display device. It seemed to us that the controlling program was going to take a great deal of time to devise without these several conveniences.

The "big boys" get around the lack of these features by purchasing [usually for $10-20,000], a Microprocessor Development System. While our observatory didn't have the ten or twenty thousand dollars to throw away, we did have access to an APPLE II computer belong-

```
;SYM AND KIM FORMAT CASSETTE TAPE OUTPUT FOR APPLE II
;
;
; LARGELY COPIED FROM THE SYNERTEK MANUAL, AND REPRODUCED
; HERE WITH THE PERMISSION OF SYNERTEK SYSTEMS CORP.
;  (STARTING AT PAGE 8 OF THE AUDIO CASSTTE INTERFACE PROGRAM)
;
;BY STEVE WELCH, 13 JUNE 79, 309 S SUNSET, LONGMONT, CO 80501,USA
; MOST SW COMMENTS ARE INDICATED BY ---
;
        .DEF    TAPOUT=$C020
;--- USE APPLE GAME PADDLE ANNUNCIATOR #0 FOR TAPE RECORDER
;---    ON-OFF CONTROL. RECORDER ON IS LOW
        .DEF    TAPEON=$C059  ;---PUT 0 HERE TO TURN ON
        .DEF    TAPEOF=$C058  ;---PUT 1 HERE TO TURN OFF
        .DEF    TM1500=$47    ;---PROB SHOULD BE TWEAKED
        .DEF    TIME99=$1A    ;---FOR DELAY ROUTINE
        .DEF    EOT=$04
        .DEF    SYN=$16
        .DEF    BUFADL=$E7    ;---ARBITRARY PLACE ON ZERO PAGE
        .DEF    BUFADH=$E8
        .DEF    CHAR=$EA
;
;---PROGRAM STARTS HERE, LINE 390 OF SYM CODE, LOC 8E87
;
        .DEF    BEGIN=$1080   ;---MUST START IN MIDDLE OF PAGE
;
        .LOC    BEGIN ;---OUT OF THE WAY OF MOST SYM PROGS
      ;--- INITILIZE
1080 20 BB11 SYMOUT: JSR   START   ;---ENTRY- PARAMETERS SET BEFORE CALL
1083 A0 80           LDY#  $80     ;---IN CASE WE TAKE KIM BRANCH
1085 2C E011         BIT   MODE    ;---TEST BIT 7 OF MODE (1=SYM,0=KIM)
1088 10 0D           BPL   DUMPT1  ;KIM-DO 128 SYNS
      ;--- WRITE 8 SECOND MARK (THIS COULD BE SHORTER)
108A A2 08           LDX#  $8      ;8 TIMES ...
108C A0 15   MARK8A: LDY#  $15     ;... ONE SEC (21 DELAYS PER SEC)
108E 20 9511 MARK8B: JSR   DELAY   ;---BENIGN PAUSE, SYM USES KIM CHAR
1091 88              DEY
1092 D0 FA           BNE   MARK8B
1094 CA              DEX
1095 D0 F5           BNE   MARK8A
      ;--- WRITE 256 SYNS, FOR SYNC
1097 A9 16   DUMPT1: LDA#  SYN
1099 20 0711         JSR   OUTCTX
109C 88              DEY
109D D0 F8           BNE   DUMPT1
      ;--- WRITE START CHARACTER
109F A9 2A           LDA#  '*'
10A1 20 0711         JSR   OUTCTX
      ;--- WRITE ID
10A4 AD DF11         LDA   ID
10a7 20 3B11         JSR   OUTBTX
```

ing to my boss, Dr. Bruce Bohannan. The APPLE has almost all of the features of the typical Microprocessor Development System, except perhaps, a means of communicating with the SBC in question. How can an APPLE talk to a SYM? Fortunately, both computers use the 6502 micro-processor chip, so programs assembled for the APPLE have little or no trouble running on the SYM or KIM. Also fortunately, all of these machines have a means of reading and writing programs on audio cassettes. It goes without saying, of course, that the tape formats of these machines are totally incompatable. So we had to do some translating; either convince the SYM to speak APPLE, or convince the APPLE to speak SYM. Since it's easier to develop programs on the APPLE [that's why I did all this in the first place], I decided to teach my APPLE to speak SYM.

It turns out that there is another good reason to teach the APPLE SYMese. The SYNERTEK people, who make the SYM, have been so kind as to publish listings of the SYM monitor in the back of their manual. This monitor listing has routines in it which produce SYM or KIM cassette tapes. The result is that the program is very easily modified to run on the APPLE. No timers are used (the APPLE has none), and the serial data is sent out through a single bit of a 6522 output port. Although the APPLE doesn't have any 6522s, it does have several single bit outputs, and in particular, it has a single bit output with the level adjusted to be used as a cassette recorder interface. Even though this is not a 6522 output, under certain conditions it can be *thought* of as one. The way that the APPLE works, any time the address of the cassette output port appears on the address bus, the cassette output flip-flop changes state. On the other hand, in the SYM, we send a particular bit pattern to an address and these bits appear on the output latch. Basically, what this means, is that we can *pretend* that the APPLE cassette is the SYM cassette output if we write only to this output when we want to *change* the level of the cassette port. With the APPLE, it should be noted, there is no control over the phase of the output signal, but all of the cassette-read routines in question are not sensitive to phase. Fortunately, through good luck or the good planning of the programmers at SYNERTEK, 90 % of the cassette output code was written in just this way. This feature makes the program a snap to adapt to the APPLE. Once I had picked out the proper pieces of the SYNERTEK code and figured out what they had done, I had only to change a few lines to obtain the results listed here. Since I did not write the program, I won't explain how it works, but I have heavily commented the listing for those readers who are interested.

Using the Program

It is a good idea to make a SYNC tape first. The APPLE output level is about ½ of the SYM's output level which may require changing the volume on playback from the usual value. Also, the APPLE does not have a high-frequency roll-off capacitor which the SYM uses, and as a result, the tone controls may need adjustment. The SYNC tape enables you to set the controls properly on your tape recorder (as outlined in the SYM manual, Appendix F). To make a SYNC tape, load the SYMOUT program into your APPLE, set the mode by setting the parameter, MODE (location $11E0), to $80 for SYM format or to $00 for KIM format and begin the program at SYNC: ($1000). This is an endless loop, so record a few minutes of the output before you hit RESET and use the resultant tape to set the level and tone on the tape recorder when reading it into the SYM (see Appendix F in SYM manual). Once you have the proper level and tone settings, down-loading your program is fairly easy. First, load the SYMOUT program. Then, load your executable program into RAM. Next, put in the parameters: Starting Address ($11DB-C), Ending Address ($11DD-E), Tape I.D. Number ($11DF), and the MODE (11E0) and start the program at SYMOUT: ($1080). Record the program, play it into your SYM, and there you have it!

Direct Computer to Computer Communication

A discovery by Dr. Bohannan: If your tape recorder has a monitor hookup, through which you can listen to whatever is being recorded, you can hook up the APPLE directly to the SYM and reduce the error rate astronomically! On our SYM (whose tape interface is modified as per MICRO's instructions), we have about a 70% chance of a successful load of our 1500 byte program with our tape recorder, a Sony. The level and tone control settings are extremely critical as well. When the machines are hooked up directly through the monitor jack of our tape recorder, we have success *every* time and the level and tone settings are unimportant. I've also found that several of my tape recorders work very well this way and have the monitor feature through the earphone jack even though it is not marked.

```
          ;--- WRITE STARTING ADDRESS
10AA AD DB11          LDA    SAL
10AD 20 3811          JSR    OUTBCX
10B0 AD DC11          LDA    SAH
10B3 20 3811          JSR    OUTBCX
10B6 2C E011          BIT    MODE     ;KIM OR HS?
10B9 10 0C            BPL    DUMPT2
          ;--- WRITE ENDING ADDRESS +1
10BB AD DD11          LDA    EAL
10BE 20 3811          JSR    OUTBCX
10C1 AD DE11          LDA    EAH
10C4 20 3811          JSR    OUTBCX
          ;--- START OF MEMORY DUMP...
          ;--- FIRST CHECK IF THIS IS THE LAST BYTE OUT
10C7 A5 E7   DUMPT2:  LDA    BUFADL   ;---LOAD ADDRESS OF CURRENT BYTE
10C9 CD DD11          CMP    EAL
10CC D0 29            BNE    DUMPT4   ;---COMPARE TO ENDING ADDRESS
10CE A5 E8            LDA    BUFADH
10D0 CD DE11          CMP    EAH
10D3 D0 22            BNE    DUMPT4   ;---BRANCH IF WE HAVE MORE TO OUTPUT
          ;--- YUP, LAST BYTE... WRITE "/"
10D5 A9 2F            LDA#   '/'
10D7 20 0711          JSR    OUTCTX
          ;--- WRITE CHECKSUM
10DA AD E111          LDA    CHKL
10DD 20 3B11          JSR    OUTBTX
10E0 AD E211          LDA    CHKH
10E3 20 3B11          JSR    OUTBTX
          ;---WRITE TWO EOT'S
10E6 A9 04            LDA#   EOT
10E8 20 3B11          JSR    OUTBTX
10EB A9 04            LDA#   EOT
10ED 20 3B11          JSR    OUTBTX
          ;--OK, NOW WE'RE DONE, SO CLEAN UP & EXIT
10F0 18               CLC             ;---INDICATE SUCESS
          ;--- SKIPPED LOTS OF STUFF, MOSTLY SYM SPECIFIC
10F1 A2 01            LDX#   $01      ;---SHUT OFF TAPE RECORDER
10F3 8E 58C0          STX    TAPEOF
10F6 60               RTS             ;---AND WE'RE ALL DONE
          ;--- NEXT IS THE CODE WHICH OUTPUTS THE NEXT MEM LOCATION
10F7 A0 00   DUMPT4:  LDY#   $0       ;---FIND THE NEXT BYTE
10F9 B1 E7            LDA@Y  BUFADL
10FB 20 3811          JSR    OUTBCX   ;WRITE IT & UPDATE CHECKSUM
10FE E6 E7            INC    BUFADL   ;BUMP BUFFER ADDR
1100 D0 C5            BNE    DUMPT2
1102 E6 E8            INC    BUFADH   ;CARRY
1104 4C C710          JMP    DUMPT2   ;---GO BACK & SEE IF WE'RE DONE
```

```
        ;--- START OF VARIOUS CHARACTER OUT ROUTINES
        ;
        ;
1107 2C E011 OUTCTX: BIT    MODE     ;HS OR KIM?
110A 10 47           BPL    OUTCHT   ;KIM TAKES BRANCH
        ;
        ;OUTBTH -NO CLOCK
        ; A,X DESTROYED
        ; MUST RESIDE ON ONE PAGE - TIMING CRITICAL
        ;
110C A2 09   OUTBTH: LDX#   $9       ;8 BITS+START BIT
110E 8C E411         STY    TEMP2
1111 85 EA           STA    CHAR
        ;---CANT READ LEVEL ON APPLE, SO NEXT INSTRUCTION IS DUMMY
1113 AD E311         LDA    TEMP1    ;--- FOR TIMING
1116 46 EA   GETBIT: LSR    CHAR
1118 49 E5           EOR#   TPBIT
111A 8D 20C0         STA    TAPOUT   ;INVERT LEVEL
        ; HERE STARTS FIRST 416 USEC PERIOD
111D A0 47           LDY#   TM1500
111F 88      A416    DEY             TIME FOR THIS LOOP IS 5Y-1
1120 D0 FD           BNE    A416
1122 90 11           BCC    NOFLIP   ;NOFLIP IF BIT 0
1124 49 E5           EOR#   TPBIT    ;BIT IS 1 - INVERT OUTPUT
1126 8D 20C0         STA    TAPOUT
        ; END OF FIRST 416 USEC PERIOD
1129 A0 46   B416:   LDY#   TM1500-1
112B 88      B416B:  DEY             ;LENGTH OF LOOP IS 5Y-1
112C D0 FD           BNE    B416B
112E CA              DEX
112F D0 E5           BNE    GETBIT   ;GET NEXT BIT (LAST IS 0 START BIT
1131 AC E411         LDY    TEMP2    ; (BY 9 BIT LSR)
1134 60              RTS
1135 EA      NOFLIP: NOP             ;TIMING
1136 90 F1           BCC    B416     ;(ALWAYS)
        ;
1138 20 AC11 OUTBCX: JSR    CHKT     ;--- GO UPDATE CHECKSUM
113B 2C E011 OUTBTX: BIT    MODE
113E 30 CC           BMI    OUTBTH   ;HS
        ;OUTBTC - OUTPUT ONE KIM BYTE
1140 A8      OUTBTC: TAY             ;SAVE DATA BYTE
1141 4A              LSRA
1142 4A              LSRA
1143 4A              LSRA
1144 4A              LSRA            ;---SHIFT HI NIBBLE INTO PLACE
1145 20 4811         JSR    HEXOUT   ; AND OUTPUT HI NIBBLE FIRST
1148 29 0F   HEXOUT: AND#   $0F      ;CONVERT LO NIBBLE TO ASCII
114A C9 0A           CMP#   $0A
114C 18              CLC
114D 30 02           BMI    HEX1
114F 69 07           ADC#   $07
1151 69 30   HEX1:   ADC#   $30
        ; OUTCHT: OUTPUTS AN ASCII CHAR IN KIM FORMAT
        ;     (MUST RESIDE ON ONE PAGE, FOR TIMING)
        ;
1153 8E E311 OUTCHT: STX    TEMP1    ;SAVE X & Y
1156 8C E411         STY    TEMP2
1159 85 EA           STA    CHAR
115B A9 FF           LDA#   $FF      ;USE FF W/SHIFTS TO COUNT BITS
115D 48      KIMBIT: PHA             ;SAVE BIT COUNTER
115E AD E411         LDA    TEMP2    ;---DUMMY FOR TIMING
1161 46 EA           LSR    CHAR     ;GET DATA BIT IN CARRY
1163 A2 12           LDX#   $12      ;ASSUME ONE
1165 B0 02           BCS    HF
1167 A2 24           LDX#   $24      ;BIT IS ZERO
1169 A0 19   HF:     LDY#   $19
116B 49 E5           EOR#   TPBIT    ;---DUMMY, REALLY
116D 8D 20C0         STA    TAPOUT   ;--- INVERT OUTPUT BIT
1170 88      HFP1:   DEY             ;PAUSE FOR 138 USEC
1171 D0 FD           BNE    HFP1
1173 CA              DEX             ;COUNT HALF CYCS OF HF
1174 D0 F3           BNE    HF
1176 A2 18   LF:     LDX#   $18      ;ASSUME BIT IS ONE
1178 B0 02           BCS    LF20
117A A2 0C           LDX#   $0C      ;BIT IS ZERO
117C A0 27   LF20:   LDY#   $27
117E 49 E5           EOR#   TPBIT    ;---DUMMY
1180 8D 20C0         STA    TAPOUT   ;---INVERT OUTPUT
1183 88      LFP1:   DEY             ;PAUSE FOR 208 USEC
1184 D0 FD           BNE    LFP1
1186 CA              DEX             ;COUNT HALF CYCS
1187 D0 F3           BNE    LF20
1189 68              PLA             ;RESTORE BIT CTR
118A 0A              ASLA            ;DECREMENT IT
118B D0 D0           BNE    KIMBIT   ;FF SHIFTED 8X=00
118D AE E311         LDX    TEMP1
1190 AC E411         LDY    TEMP2
1193 98              TYA             ;RESTORE X,Y,DATA BYTE
1194 60              RTS
        ;--- WE NEED A DELAY FUNCTION, BECAUSE THE SYM PROG USES THE KIM
        ;    CHAROUT ROUTINE WITH OUTPUT DISABLED TO DELAY (& WE CAN'T)
        ;
        ;--- THIS ONE SHOULD BE 1/21 SEC , SINCE IT EMULATES
        ;--- THE KIM CHAR OUT ROUTINE, WHICH THE SYM PROGRAM USES
1195 8E E311 DELAY:  STX    TEMP1    ;---PRESERVE X
1198 8C E411         STY    TEMP2    ;--- AND Y
119B A2 00           LDX#   $00      ;---DO OUTER LOOP 256 TIMES
119D A0 1A   LOOP0:  LDY#   TIME99   ;---LOOP
119F 88      LOOP1:  DEY
11A0 D0 FD           BNE    LOOP1
11A2 CA              DEX
11A3 D0 F8           BNE    LOOP0
11A5 AE E311         LDX    TEMP1    ;---RESTORE X
11A8 AC E411         LDY    TEMP2    ;--- AND Y
11AB 60              RTS
        ;
        ;--- CHKT... UPDATE CHECKSUM FROM BYTE IN ACC
11AC A8      CHKT:   TAY             ;SAVE ACC
11AD 18              CLC
11AE 6D E111         ADC    CHKL
11B1 8D E111         STA    CHKL
11B4 90 03           BCC    CHKT10
11B6 EE E211         INC    CHKH     ;BUMP HI BYTE
11B9 98      CHKT10: TYA             ;RESTORE ACC
11BA 60              RTS
        ;--- START... LEAVING OUT SOME UNECESSARY JUNCK
11BB 20 C711 START:  JSR    ZERCK    ;ZERO CHECKSUM
11BE 20 D011         JSR    P2SCR    ;---THATS WHAT THEY NAMED IT
11C1 A9 00           LDA#   $00      ;---TURN ON TAPE RECORDER
11C3 8D S9C0         STA    TAPEON
11C6 60              RTS
11C7 A9 00   ZERCK:  LDA#   $00      ;ZERO CHECKSUM
11C9 8D E111         STA    CHKL
11CC 8D E211         STA    CHKH
11CF 60              RTS
        ;--- P2SCR... THIS MOVES THE STARTING ADDRESS TO THE
        ;            RUNNING BUFFER ADDRESS. THE WEIRD NAME
        ;            IS DUE TO THE NAMES OF THE LOCATIONS
        ;            WHICH WE ARE MOVING IN THE SYM BOOK
```

```
11D0 AD DC11 P2SCR:  LDA    SAH      ;---STARTING ADD HI
11D3 85 E8           STA    BUFADH
11D5 AD DB11         LDA    SAL      ;---STARTING ADD LO
11D8 85 E7           STA    BUFADL
11DA 60              RTS

                .PAGE
          ;
          ;
          ;--- PARAMETERS, ETC.
          ;
          ;--- THESE NEXT SIX LOCATIONS SHOULD BE FILLED WITH THE CALLING
          ;    PARAMETERS BEFORE CALLING THE SYMOUT ROUTINE
          ;
11DB 00         SAL:    .BYTE ;---STARTING ADDRESS, LO BYTE
11DC 00         SAH:    .BYTE ;---     "        "   , HI BYTE
11DD 00         EAL:    .BYTE ;---ENDING ADDRESS +1, LO BYTE
11DE 00         EAH:    .BYTE ;---     "        "      , HI BYTE
11DF 00         ID:     .BYTE ;---TAPE ID NUMBER
11E0 00         MODE:   .BYTE ;---SYM=$80, KIM=$00
          ;
11E1 00         CHKL:   .BYTE ;---VARIABLES
11E2 00         CHKH:   .BYTE
11E3 00         TEMP1:  .BYTE
11E4 00         TEMP2:  .BYTE
11E5 00         TPBIT:  .BYTE
          ;
          ;
          ;--- A SHORT ROUTINE TO MAKE SYNC TAPES...
          ;    (THE APPLE PRODUCED TAPE WILL USUALLY NEED DIFFERENT
          ;     VOLUME & TONE SETTING THAN SYM OR KIM TAPES)

                .LOC    $1000
1000 20 BB11 SYNC:   JSR    START    ;---MAKE A SYNC TAPE
1003 A9 16   SYNMOR: LDA#   SYN      ;---LOAD SYNC CHAR
1005 20 0711         JSR    OUTCTX   ;---SEND IT
1008 4C 0310         JMP    SYNMOR   ;---DO IT FOREVER
          ;
          ;
                .END

.       100B      A416    111F      B416    1129
B416B   112B      BEGIN   1080      BUFADH  00E8
BUFADL  00E7      CHAR    00EA      CHKH    11E2
CHKL    11E1      CHKT    11AC      CHKT10  11B9
DELAY   1195      DUMPT1  1097      DUMPT2  10C7
DUMPT4  10F7      EAH     11DE      EAL     11DD
EOT     0004      GETBIT  1116      HEX1    1151
HEXOUT  1148      HF      1169      HFP1    1170
ID      11DF      KIMBIT  115D      LF      1176
LF20    117C      LFP1    1163      LOOP0   119D
LOOP1   119F      MARK8A  108C      MARK8B  108E
MODE    11E0      NOFLIP  1135      OUTBCX  1138
OUTBTC  1140      OUTBTH  110C      OUTBTX  113B
OUTCHT  1153      OUTCTX  1107      P2SCR   11D0
SAH     11DC      SAL     11DB      START   11BB
SYMOUT  1060      SYN     0016      SYNC    1000
SYNMOR  1003      TAPEOF  C058      TAPEON  C059
TAPOUT  C020      TEMP1   11E3      TEMP2   11E4
TIME99  001A      TM1500  0047      TPBIT   11E5
ZERCK   11C7

A416    111F 1120
B416    1129 1136
B416B   112B 112C
BEGIN   0000 0000
BUFADH  0000 10CE 1102 11D3
BUFADL  0000 10C7 10F9 10FE 11D8
CHAR    0000 1111 1116 1159 1161
CHKH    10E0 11B6 11CC 11E2
CHKL    10DA 11AE 11B1 11C9 11E1
CHKT    1138 11AC
CHKT10  11B4 11B9
DELAY   108E 1195
DUMPT1  1088 1097 109D
DUMPT2  10B9 10C7 1100 1104
DUMPT4  10CC 10D3 10F7
EAH     10C1 10D0 11DE
EAL     10BB 10C9 11DD
EOT     0000 10E6 10EB
GETBIT  1116 112F
HEX1    114D 1151
HEXOUT  1145 1148
HF      1165 1169 1174
HFP1    1170 1171
ID      10A4 11DF
KIMBIT  115D 118B
LF      1176
LF20    1178 117C 1187
LFP1    1183 1184
LOOP0   119D 11A3
LOOP1   119F 11A0
MARK8A  108C 1095
MARK8B  108E 1092
MODE    1085 10B6 1107 113B 11E0
NOFLIP  1122 1135
OUTBCX  10AD 10B3 10BE 10C4 10FB 1138
OUTBTC  1140
OUTBTH  110C 113E
OUTBTX  10A7 10DD 10E3 10E8 10ED 113B
OUTCHT  110A 1153
OUTCTX  1099 10A1 10D7 1107 1005
P2SCR   11BE 11D0
SAH     10B0 11D0 11DC
SAL     10AA 11D5 11DB
START   1060 11BB 1000
SYMOUT  1060
SYN     0000 1097 1003
SYNC    1000
SYNMOR  1003 1008
TAPEOF  0000 10F3
TAPEON  0000 11C3
TAPOUT  0000 111A 1126 116D 1180
TEMP1   1113 1153 118D 1195 11A5 11E3
TEMP2   110E 1131 1156 115E 1190 1198 11A8 11E4
TIME99  0000 119D
TM1500  0000 111D 1129
TPBIT   1118 1124 116B 117E 11E5
ZERCK   11BB 11C7

NO ERRORS DETECTED
```

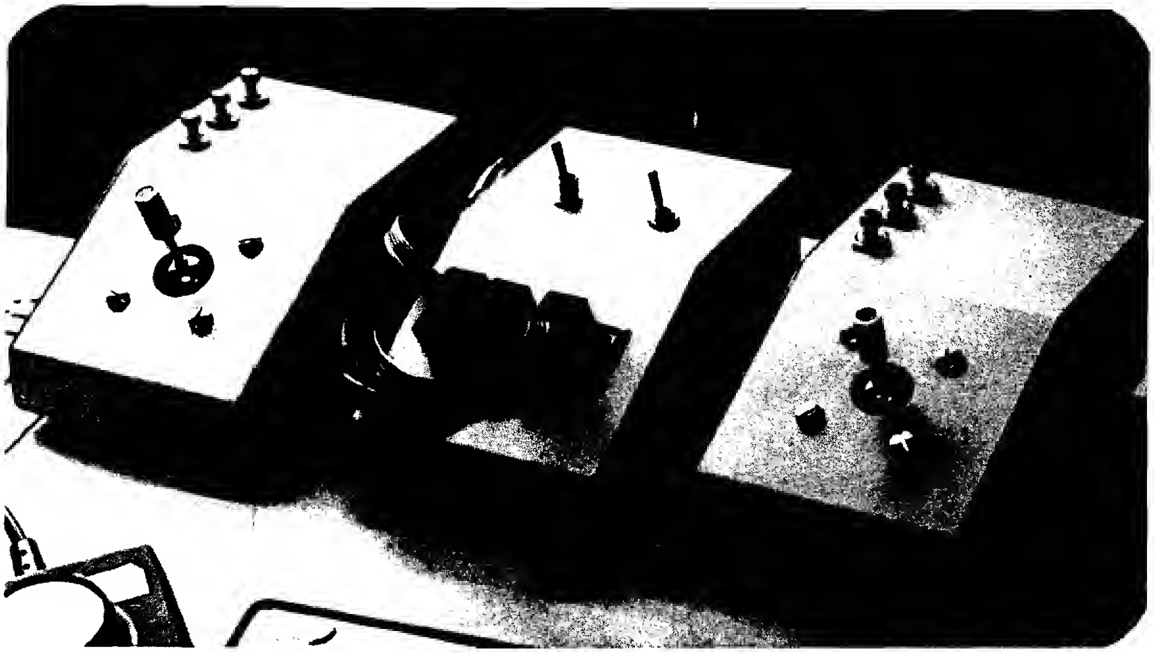
```
                 ;               APPLE GLASS TTY PROGRAM
                 ;
                 ; TERMINAL PROGRAM - MAKES APPLE LOOK LIKE A STUPID TERMINAL
                 ;     BY STEVE WELCH,  14 AUG 1979
                 ;      PHONE (303)-776-3414
                 ;       309 S SUNSET, LONGMONT, CO 80501, USA
                 ;
                 ; FIRST SOME DEFINITIONS...
                 ;
                 ;  THIS PROGRAM USES ELECTRONIC SYSTEMS SERIAL I/O BOARD
                      .DEF    SLOT=$40       ;(10*SLOT THAT BOARD IS IN)
                      .DEF    STATUS=$C081+SLOT
                      .DEF    INPUT =$C080+SLOT
                      .DEF    OUTPUT=$C082+SLOT

                      .DEF    COUT=$FDF0     ;APPLE MONITOR CHAR OUT
                      .DEF    HOME=$FC58     ;CLEAR SCR &HOME CURSOR
                      .DEF    BASL=$28       ;CURSOR  POSITION (SORTA)
                      .DEF    CH=$24         ;CURSOR HORIZONTAL POS.
                      .DEF    KEYBD=$C000    ;KEYBOARD DATA
                      .DEF    KEYSTB=$C010   ;KEYBOARD STROBE CLEAR
                      .DEF    PDL0=$C061     ;APPLE PADDLE SWITCH ZERO
                      .DEF    PDL1=$C062     ; "      "      "   ONE
                      .DEF    BEEP=$FF3A

                      .DEF    POINT=$F0      ;ACTUAL BUFFER POINTER
                      .DEF    POINTL=POINT   ; "     "     "  LO BYTE
                      .DEF    POINTH=$F1     ; "     "     "  HI BYTE
                 ;
                      .LOC    $1000
                 ;
                 ;--- INITILIZE, HOME & START CURSOR
                 ;
1000 A9 00                LDA#   $00      ;SET BEGINNING OF BUFFER TO $2000
1002 8D F410              STA    BUFMTL
1005 85 F0                STA    POINTL   ; AND SET BUFFER EMPTY
1007 A9 20                LDA#   $20
1009 8D F510              STA    BUFMTH
100C 85 F1                STA    POINTH
                 ;
100E 20 58FC HOMEIT:      JSR    HOME
1011 A9 60                LDA#   $60      ;PUT FLASHING CURSOR ON SCREEN
1013 A4 24                LDY    CH       ; IN HOME POSITION
1015 91 28                STA@Y  BASL
                 ;
                 ;--- MAIN LOOP OF PROGRAM
                 ;
1017 20 3210 MAIN:        JSR    GETKEY   ;GET KEY PRESSED, STICK IN BUFFER
101A 20 2310              JSR    GETCHR   ;GET CHAR FROM INPUT, PUT ON SCREEN
101D 20 4E10              JSR    CHKBUF   ;OUTPUT NEXT CHAR FROM BUFFER
1020 4C 1710              JMP    MAIN
                 ;--- GETCHR: GETS ASCII CHARACTER FROM INPUT PORT
                 ;            AND PUTS IT ON SCREEN.
                 ;
                 ;--- CALLED BY MAIN LOOP
1023 AD C1C0 GETCHR:      LDA    STATUS   ;FIRST CHECK HAS DATA COME IN
1026 29 80                AND#   $80      ;MASK OFF DATA AVAILABLE
1028 D0 01                BNE    GOON     ;TAKES BRANCH, GOT SOMETHING
102A 60                   RTS
102B AD C0C0 GOON:        LDA    INPUT    ;GET CHAR FROM BOARD
102E 20 7D10              JSR    CHROUT   ;SEND IT TO TV
1031 60                   RTS
                 ;
                 ;--- GETKEY: IF KEY IS PRESSED, GETS KEY, CONVERTS TO ASCII,
                 ;            AND STICKS IN BUFFER.
                 ;
                 ;--- CALLED BY MAIN LOOP
1032 2C 00C0 GETKEY:      BIT    KEYBD    ;BIT 7 SET IF KEY PRESSED
1035 30 01                BMI    GOON2    ;TAKES BRANCH IF KEY PRESSED
1037 60                   RTS
1038 AD 00C0 GOON2:       LDA    KEYBD    ;GET CHAR
103B 29 7F                AND#   $7F      ;CONVERT TO ASCII
103D 2C 10C0              BIT    KEYSTB   ;CLEAR KEYBOARD STROBE
1040 20 9810              JSR    CHKSPC   ;GO CHECK FOR SPECIAL CHARS
1043 A0 00                LDY#   $00      ;NOW STICK THIS CHAR IN BUFFER
1045 E6 F0                INC    POINTL   ;INCREMENT POINTER
1047 D0 02                BNE    SAVEIT   ;CHECK FOR CARRY
1049 E6 F1                INC    POINTH   ;INCREMENT HI BYTE
104B 91 F0   SAVEIT:      STA@Y  POINT    ;STICK IT
104D 60                   RTS
                 ;
                 ;--- CHKBUF: CHECK BUFFER, AND IF THERE IS SOMETHING IN IT,
                 ;            SEND IT OUT TO THE OUTPUT PORT. (ECHO IF ON)
                 ;
                 ;--- CALLED BY MAIN LOOP
104E A5 F0   CHKBUF:      LDA    POINTL   ;FIRST LETS SEE IF BUFFER EMPTY
1050 CD F410              CMP    BUFMTL   ;CHECK- IF POINT .NE. BUFMT,
1053 D0 07                BNE    NOTMT    ; THEN GO OUTPUT THE NEXT CHAR
1055 A5 F1                LDA    POINTH   ;CHECK HI BYTE
1057 CD F510              CMP    BUFMTH
105A F0 20                BEQ    BUFDUN   ;TAKE BRANCH IF BUFFER IS EMPTY
105C A0 00   NOTMT:       LDY#   $00
105E AD C1C0 WAIT:        LDA    STATUS   ;NOW WAIT TILL BOARD IS READY
1061 29 01                AND#   $01      ;(TBR BIT)
1063 F0 F9                BEQ    WAIT
1065 B1 F0                LDA@Y  POINT    ;FALLS THRU WHEN BOARD READY-GET CHAR
1067 8D C2C0              STA    OUTPUT   ;SEND TO OUTPUT PORT
106A 2C F210              BIT    ECHO     ;TEST ECHO ON ?
106D 10 03                BPL    NOECHO
106F 20 7D10              JSR    CHROUT   ;YES, SEND CHAR TO TV
1072 C6 F0   NOECHO:      DEC    POINTL   ;NOW DECREMENT BUFFER POINTER
1074 A5 F0                LDA    POINTL
1076 C9 FF                CMP#   $FF      ;TEST FOR BORROW
1078 D0 02                BNE    BUFDUN
107A C6 F1                DEC    POINTH   ;DECREMENT HI BYTE
107C 60      BUFDUN:      RTS

*** ASM65 ***   (V2A)   MAR-79


                          .PAGE
                 ;--- CHROUT: SENDS ASCII CHARACTER IN ACCUMULATOR TO SCREEN
                 ;            ALSO KEEPS TRACK OF CURSOR, AND CHANGES ASCII
                 ;            LOWER CASE TO REVERSE VIDEO.
                 ;
                 ;---CALLED BY GETCHR, AND (WHEN ECHO ON) CHKBUF
107D 48      CHROUT:      PHA             ;SAVE CHAR...
107E A4 24                LDY    CH       ;FIRST RESTORE CURSOR
1080 B1 28                LDA@Y  BASL
1082 49 C0                EOR#   $C0      ;UNFLASH IT, SORTA
1084 91 28                STA@Y  BASL     ; AND PUT IT BACK
1086 68                   PLA             ;GET THE CHAR BACK
1087 09 80                ORA#   $80      ;FIX FOR APPLE SCREEN
1089 20 CB10              JSR    CHKLC    ;CHECK & FIX LOWER CASE TO REVERSE VID
108C 20 F0FD              JSR    COUT     ;PUT CHARACTER ON SCREEN
108F A4 24                LDY    CH       ;PUT UP NEW CURSOR...
1091 B1 28                LDA@Y  BASL
1093 49 C0                EOR#   $C0      ;MAKE IT A FLASHING WHATEVER
1095 91 28                STA@Y  BASL
1097 60                   RTS
```

```
                 ;
                 ;--- CHKSPC: CHECK CHARACTERS FROM KEYBOARD FOR SPECIAL CURSOR
                 ;              CONTROL CHARACTERS, AND LOWER CASE.
                 ;
                 ;--- CALLED BY GETKEY
1098 48               CHKSPC: PHA              ;SAVE THE CHARACTER
1099 C9 15                    CMP#   $15       ;FWD ARROW
109B F0 10                    BEQ    CTRLU
109D 2C 61C0                  BIT    PDL0      ;IF PDL 0 IS PRESSED, THIS CHAR IS
10A0 10 04                    BPL    SPC2      ; A SETUP COMMAND
10A2 68                       PLA
10A3 20 D810                  JSR    SETUP     ;KEY IN ACC
10A6 2C 62C0  SPC2:           BIT    PDL1      ;CHECK PADDLE "UNSHIFT KEY"
10A9 30 1C                    BMI    LCASE     ;TAKES BRANCH IF PADDLE #1 PRESSED
10AB 68                       PLA
10AC 60                       RTS
                 ;
10AD 68               CTRLU:  PLA              ;DISCARD CTRL U
10AE A4 24                    LDY    CH        ;SEND OUT WHATS UNDER THE CURSOR
10B0 B1 28                    LDA@Y  BASL
10B2 49 C0                    EOR#   $C0       ;UNFLASH IT
10B4 8D F110                  STA    TEMP1     ;SAVE IT, WHILE WE CHECK FOR REVERSE
10B7 29 E0                    AND#   $E0       ;MASK OFF REVERSE BITS
10B9 F0 06                    BEQ    REV       ;...THEY ARE ALL ZERO FOR REVERSE
10BB AD F110                  LDA    TEMP1     ;NOT REVERSE- GET OUR CHAR BACK
10BE 29 7F                    AND#   $7F       ;CONVERT TO ASCII
10C0 60                       RTS              ;AND RETURN
                 ;
10C1 AD F110  REV:            LDA    TEMP1     ;CHANGE THE REVERSED CHAR TO ASCII LC
10C4 09 60                    ORA#   $60
10C6 60                       RTS              ;AND RETURN...
                 ;
10C7 68               LCASE:  PLA              ;MAKE THE KEYBOARD SEND OUT LOWER CASE
10C8 09 60                    ORA#   $60       ;ASCII LOWER CASE
10CA 60                       RTS              ;AND RETURN WITH IT IN ACC
                 ;---CHKLC:  CHECK INCOMING CHARACTERS FOR LOWER CASE AND
                 ;            IF THEY ARE MAKE THEM REVERSE VIDEO, AS THE
                 ;            APPLE DOESNT HAVE A LOWER CASE CHAR. GEN.
                 ;
                 ;--- CALLED BY CHROUT
10CB 48               CHKLC:  PHA              ;SAVE SO WE CAN CHECK IT
10CC 29 60                    AND#   $60       ;MASK IT OFF
10CE C9 60                    CMP#   $60
10D0 D0 04                    BNE    NOTLC     ;TAKE BRANCH IF NOT LOWER CASE
10D2 68                       PLA              ;ITS LOWER CASE,
10D3 29 1F                    AND#   $1F       ;SO MAKE IT REVERSE VIDEO
10D5 60                       RTS              ; AND RETURN
10D6 68               NOTLC:  PLA
10D7 60                       RTS
                 ;
                 ;--- SETUP:  CHANGES THE CHARACTERISTICS OF THE APPLE TERMINAL.
                 ;            WHEN THE PADDLE 0 PUSH BUTTON IS PRESSED ALONG
                 ;            WITH A KEY THIS KEY IS A MACHINE COMMAND.
                 ;
                 ;--- CALLED BY CHKSPC
10D8 C9 45     SETUP:         CMP#   $45       ;CHECK FOR "E" (ECHO)
10DA F0 03                    BEQ    ECHO1
                 ; INSERT OTHER TESTS HERE
10DC 4C EA10                  JMP    SETOUT    ;NONE OF THE ABOVE
10DF AD F210  ECHO1:          LDA    ECHO      ;FLIP ECHO MODE
10E2 49 80                    EOR#   $80
10E4 8D F210                  STA    ECHO
10E7 4C EA10                  JMP    SETOUT
10EA 68               SETOUT: PLA              ;RETURN TO MAIN LOOP
10EB 68                       PLA              ; (THIS ISNT REALLY A SUBROUTINE)
10EC 68                       PLA
10ED 68                       PLA              ;PULL THE RETURN ADD.'S OFF THE STACK
10EE 4C 1710                  JMP    MAIN
                 ;
10F1 00          TEMP1:   .BYTE
10F2 80          ECHO:    .BYTE $80          ;$80=ECHO ON
10F3 E0          REVERS:  .BYTE $E0          ;REVERS VIDEO MASK
10F4 00          BUFMTL:  .BYTE              ;FIRST BYTE IN BUFFER
10F5 00          BUFMTH:  .BYTE              ; DITTO, HI BYTE
                 .END
```

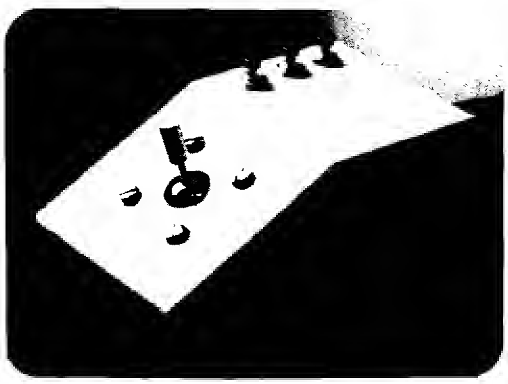
```
.       10F6          BASL    0028          BEEP    FF3A
BUFDUN  107C          BUFMTH  10F5          BUFMTL  10F4
CH      0024          CHKBUF  104E          CHKLC   10CB
CHKSPC  1098          CHROUT  107D          COUT    FDF0
CTRLU   10AD          ECHO    10F2          ECHO1   10DF
GETCHR  1023          GETKEY  1032          GOON    102B
GOON2   1038          HOME    FC58          HOMEIT  100E
INPUT   C0C0          KEYBD   C000          KEYSTB  C010
LCASE   10C7          MAIN    1017          NOECHO  1072
NOTLC   10D6          NOTMT   105C          OUTPUT  C0C2
PDL0    C061          PDL1    C062          POINT   00F0
POINTH  00F1          POINTL  00F0          REV     10C1
REVERS  10F3          SAVEIT  104B          SETOUT  10EA
SETUP   10D8          SLOT    0040          SPC2    10A6
STATUS  C0C1          TEMP1   10F1          WAIT    105E
BASL    0000 1015 1080 1084 1091 1095 10B0
BEEP    0000
BUFDUN  105A 1078 107C
BUFMTH  1009 1057 10F5
BUFMTL  1002 1050 10F4
CH      0000 1013 107E 108F 10AE
CHKBUF  101D 104E
CHKLC   1089 10CB
CHKSPC  1040 1098
CHROUT  102E 106F 107D
COUT    0000 108C
CTRLU   109B 10AD
ECHO    106A 10DF 10E4 10F2
ECHO1   10DA 10DF
GETCHR  101A 1023
GETKEY  1017 1032
GOON    1028 102B
GOON2   1035 1038
HOME    0000 100E
HOMEIT  100E
INPUT   0000 102B
KEYBD   0000 1032 1038
KEYSTB  0000 103D
LCASE   10A9 10C7
MAIN    1017 1020 10EE
NOECHO  106D 1072
NOTLC   10D0 10D6
NOTMT   1053 105C
OUTPUT  0000 1067
PDL0    0000 109D
PDL1    0000 10A6
POINT   0000 0000 104B 1065
POINTH  0000 100C 1049 1055 107A
POINTL  0000 1005 1045 104E 1072 1074
REV     10B9 10C1
REVERS  10F3
SAVEIT  1047 104B
SETOUT  10DC 10E7 10EA
SETUP   10A3 10D8
SLOT    0000 0000 0000 0000
SPC2    10A0 10A6
STATUS  0000 1023 105E
TEMP1   10B4 10BB 10C1 10F1
WAIT    105E 1063
```

APPLE II® JOYSTICK & EXPANDA·PORT

PROGRAMMA

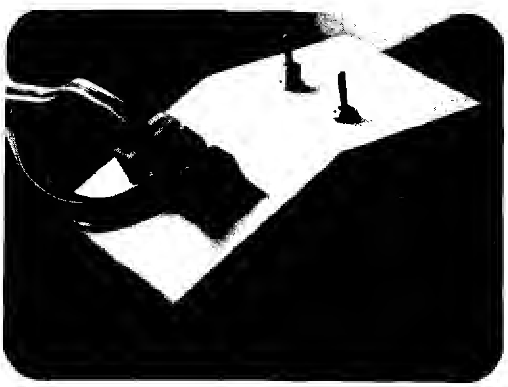


EVERY APPLE II OWNER SHOULD HAVE ONE!



JOYSTICK $49.95

The PROGRAMMA JOYSTICK is an input peripheral that attaches to the APPLE II Computer's game I/O Port. The JOYSTICK is a must for the serious game player, and it offers a degree of linearity not currently available with other joysticks. The ease of manueverability and the availability of the "functional" switches make the PROGRAMMA JOYSTICK a much needed enhancement to any APPLE II Computer System owner. The PROGRAMMA JOYSTICK comes completely assembled and tested, including a User's Guide.



EXPANDA-PORT $49.95

The PROGRAMMA EXPANDA-PORT is a multi-port expander for the game I/O port of any APPLE II Computer System. In addition to allowing expansion for up to six devices, the EXPANDA-PORT contains a built-in speaker that replaces the function of the Apple II's speaker. The switches on the EXPANDA-PORT allow for the selection of the specific device desired and for the switching of that device. No unplugging of any device connected to the EXPANDA-PORT is required. The PROGRAMMA EXPANDA PORT comes completely assembled and tested, including a User's guide.

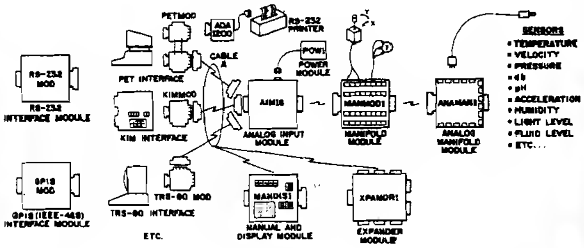
The PROGRAMMA JOYSTICK and EXPANDA-PORT are available on a limited basis through your local computer dealer. Apple II is a registered trademark of Apple Computers, Inc.

Computer System Products

PROGRAMMA INTERNATIONAL, INC.

3400 Wilshire Blvd.
Los Angeles, CA 90010 (213) 384-0579 · 384-1116 · 384-1117

Data Acquisition Modules

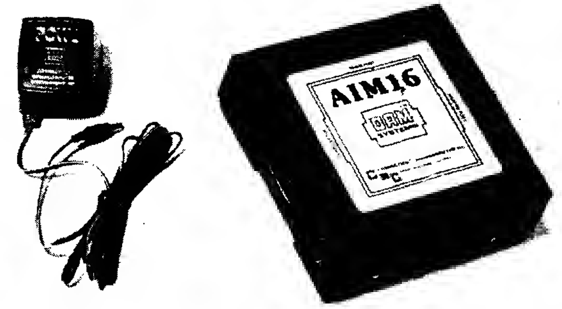


The world we live in is full of variables we want to measure. These include weight, temperature, pressure, humidity, speed and fluid level. These variables are continuous and their values may be represented by a voltage. This voltage is the analog of the physical variable. A device which converts a physical, mechanical or chemical quantity to a voltage is called a sensor.

Computers do not understand voltages: They understand bits. Bits are digital signals. A device which converts voltages to bits is an analog-to-digital converter. Our AIM16 (Analog Input Module) is a 16 input analog-to-digital converter.

The goal of Connecticut microComputer in designing the DAM SYSTEMS is to produce easy to use, low cost data acquisition modules for small computers. As the line grows we will add control modules to the system. These acquisition and control modules will include digital input sensing (e.g. switches), analog input sensing (e.g. temperature, humidity), digital output control (e.g. lamps, motors, alarms), and analog output control (e.g. X-Y plotters, or oscilloscopes).

Analog Input Module



The AIM16 is a 16 channel analog to digital converter designed to work with most microcomputers. The AIM16 is connected to the host computer through the computer's 8 bit input port and 8 bit output port, or through one of the DAM SYSTEMS special interfaces.

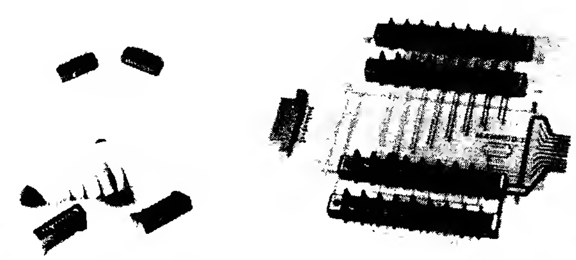
The input voltage range is 0 to 5.12 volts. The input voltage is converted to a count between 0 and 255 (00 and FF hex). Resolution is 20 millivolts per count. Accuracy is 0.5% ± 1 bit. Conversion time is less than 100 microseconds per channel. All 16 channels can be scanned in less than 1.5 milliseconds.

Power requirements are 12 volts DC at 60 ma.

The POW1 is the power module for the AIM16. One POW1 supplies enough power for one AIM16, one MANMOD1, sixteen sensors, one XPANDR1 and one computer interface. The POW1 comes in an American version (POW1a) for 110 VAC and in a European version (POW1e) for 230 VAC.

AIM16 . . . $179.00
POW1a . . . $ 14.95
POW1e . . . $ 24.95

Connectors



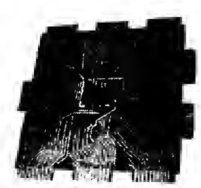
The AIM16 requires connections to its input port (analog inputs) and its output port (computer interface). The ICON (Input CONnector) is a 20 pin, solder eyelet, edge connector for connecting inputs to each of the AIM16's 16 channels. The OCON (Output CONnector) is a 20 pin, solder eyelet edge connector for connecting the computer's input and output ports to the AIM16.

The MANMOD1 (MANifold MODule) replaces the ICON. It has screw terminals and barrier strips for all 16 inputs for connecting pots, joysticks, voltage sources, etc.

CABLE A24 (24 inch interconnect cable has an interface connector on one end and an OCON equivalent on the other. This cable provides connections between the DAM SYSTEMS computer interfaces and the AIM16 or XPANDR1 and between the XPANDR1 and up to eight AIM16s.

ICON . . . $ 9.95
OCON . . . $ 9.95
MANMOD1 . . . $59.95
CABLE A24 . . . $19.95

XPANDR1



The XPANDR1 allows up to eight AIM16 modules to be connected to a computer at one time. The XPANDR1 is connected to the computer in place of the AIM16. Up to eight AIM16 modules are then connected to each of the eight ports provided using a CABLE A24 for each module. Power for the XPANDR1 is derived from the AIM16 connected to the first port.

XPANDR1 . . . $59.95

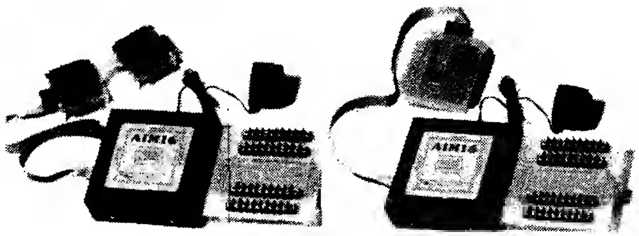
TEMPSENS

This module provides two temperature probes for use by the AIM16. This module should be used with the MANMOD1 for ease of hookup. The MANMOD1 will support up to 16 probes (eight TEMPSENS modules).

Resolution for each probe is 1°F.

TEMPSENS2P1 (-10°F to 120°F) . . . $49.95

Computer Interfaces and Sets



For your convenience the AIM16 comes as part of a number of sets. The minimum configuration for a usable system is the AIM16 Starter Set 1. This set includes one AIM16, one POW1, one ICON and one OCON. The AIM16 Starter Set 2 includes a MANMOD1 in place of the ICON. Both of these sets require that you have a hardware knowledge of your computer and of computer interfacing.

For simple plug compatible systems we also offer computer interfaces and sets for several home computers.

The PETMOD plugs into the back of the Commodore PET computer and provides two PET IEEE ports, one user port and one DAM SYSTEMS port. The PETMOD is connected to the AIM16 or XPANDR1 with CABLE A24. The PETSET1 includes one PETMOD, one CABLE A24, one AIM16, one POW1 and one MANMOD1. To read and display a single AIM16 channel (N) using the PETSET1 the BASIC statements

```
POKE59426,N:POKE59426,255:X=PEEK(59471):PRINT"CHANNEL "N"="X
```

are all that is needed.

The KIMMOD plugs into the COMMODORE KIM applications connector and provides one application connector and one DAM SYSTEM'S port. The KIMMOD is connected to the AIM16 or XPANDR1 with CABLE A24. Assembly and machine language programs for reading and displaying data are included. The KIMSET1 includes one KIMMOD, one CABLE A24, one AIM16, one POW1 and one MANMOD 1.

All sets come in American and European versions.

AIM16 Starter Set 1a (110 VAC) . . . $189.00
AIM16 Starter Set 1e (230 VAC) . . . $199.00
AIM16 Starter Set 2a (110 VAC) . . . $259.00
AIM16 Starter Set 2e (230 VAC) . . . $269.00
PETMOD . . . $ 49.95
KIMMOD . . . $ 39.95
PETSET 1a . . . $295.00
PETSET1e . . . $305.00
KIMSET1a . . . $285.00
KIMSET1e . . . $295.00

Coming Soon

TEMPSENS-2P with other temperature ranges. Interfaces for TRS-80, APPLE, AIM65. Light sensors. Output modules. Contact us for price and availability.

Dealers

Give your potential customers a reason for buying your computers. We offer excellent discounts to legitimate dealers. Contact us for our dealer pack.

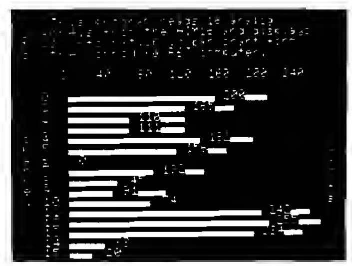
Our Guarantee of Satisfaction

Our customers are our most important asset. We want you to be pleased with whatever you purchase from us. We strive to offer top quality products at reasonable prices. We believe you should see an item before you spend your hard earned cash for it.

Ask for a demonstration at your local computer store so you can be sure our products perform as you want them to perform. Your dealer is a valued source of information and advice.

If you cannot see our products in advance, and order direct from us, we offer a money back guarantee. If our products don't perform as you expect, return the merchandise to us within 30 days, in its original condition, and we will refund the purchase price.

Our standard warranty for all our products is 90 days.



Order Form

CONNECTICUT microCOMPUTER, Inc.
150 POCONO ROAD
BROOKFIELD, CONNECTICUT 06804
TEL: (203) 775-9659 TWX: 710-456-0052

QUANTITY	DESCRIPTION	PRICE	TOTAL
SUBTOTAL			
Handling and shipping — add per order			$3.00
Foreign orders add 10% for AIR postage			
Connecticut residents add 7% sales tax			
TOTAL ENCLOSED			

NAME ____________________

COMPANY ____________________

ADDRESS ____________________

____

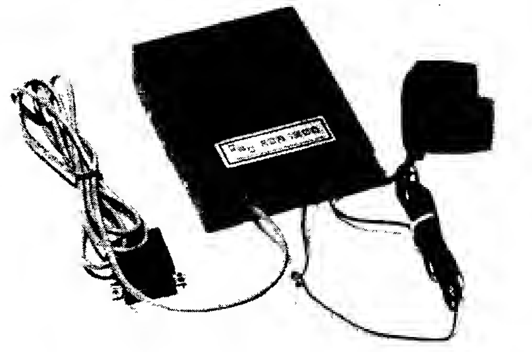
CITY ____________________

STATE ____________________ ZIP ________

VISA ○ M/C ○ Expiration date ____________________

Card number ____________________

PET Printer Adapter



The CmC ADA 1200 drives an RS-232 printer from the PET IEEE-488 bus. Now, the PET owner can obtain hard copy listings and can type letters, manuscripts, mailing labels, tables of data, pictures, invoices, graphs, checks, needlepoint patterns, etc., using RS-232 standard printer or terminal.

A cassette tape is included with software for plots, formatting tables and screen dumps. The ADA1200 sells for $169.00 and includes case, power supply and cable.

Order direct or contact your local computer store.

VISA AND M/C ACCEPTED — SEND ACCOUNT NUMBER, EXPIRATION DATE AND SIGN ORDER.
ADD $3 PER ORDER FOR SHIPPING & HANDLING — FOREIGN ORDERS ADD 10% FOR AIR POSTAGE

CONNECTICUT microCOMPUTER, Inc.
150 POCONO ROAD
BROOKFIELD, CONNECTICUT 06804
TEL: (203) 775-9659 TWX: 710-456-0052

Apple-Doc

By Roger Wagner

An Aid to the Development and Documentation of Applesoft Programs

This 3 program set is a must to anyone writing or using programs in Applesoft! It not only provides valuable info. on each of your programs, but allows you to change any element throughout the listing almost as easily as you would change a single line!!

With Apple-Doc you can produce a list of every variable in your program and the lines each is used on, each line called by a GOTO, GOSUB, etc., in fact, every occurance of almost anything!

You can rename variables, change constants and referenced line #'s, or do local or global replacement editing on your listing.

In fact, we guarantee that after purchase, if you don't feel APPLE-DOC is one of the most valuable programs in your library we will even refund your money! (Upon return of product.)

Unheard of? Yes! But that's how good APPLE-DOC really is!

That's not all!! Send for free info. or visit your nearest Apple dealer.

Only $19.95 **Please specify diskette or tape.**
(Calif. residents add 6% Sales Tax)

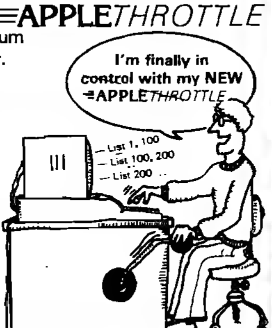
Available from your local computer store or:

Southwestern Data Systems
P.O. Box 582-M
Santee, CA 92071
(714) 562-3670

(Dealer inquiries invited)

Put Yourself in Control with the APPLE*THROTTLE*

That's right! The APPLE*THROTTLE* will turn your game paddles into a speed controller. By simply pushing a button, you can stop your computer for as long as you want. Release the button, and your computer enters a slow-motion mode with one paddle controlling the speed. And if that isn't enough, look at these additional features:



- Plugs into any slot
- Works with machine language, Integer BASIC, and Applesoft
- Normal · slow · stop
- Use to LIST, TRACE, RUN, etc.
- NO SOFTWARE to load
- Unveil program secrets

APPLE*THROTTLE* **$89.95**

And there's more! No more multiple LIST commands to view small program sections. With the APPLE*THROTTLE*, you'll be able to list or trace long programs while watching your program flow in slow-motion. So get in control with the APPLE*THROTTLE* and order yours today!

APPLETIME, a Real Time Clock for the Apple II. Plugs directly into any slot and keeps time even when computer is off. Features 12/24 Hour, BCD/ASCII data format, and AC/Crystal time base selection. Includes software examples for machine language and BASIC programs. Completely assembled and tested.
APT-1 Real Time Clock **$79.95**

PROTOBOARD, with over 1300 holes on 0.1 centers for designing your own circuits.
APB-1 Protoboard **$17.95**

VERBATIM 5¼" DISKETTES
Soft-Sector Box of 10 ... **$34.50**
(plastic file case included)

west side electronics
P.O. Box 636, Chatsworth, CA 91311
We pay all shipping in Continental U.S.A.
Others add 10% California residents add 6% tax

PET Word Processor

8K and 16/32K PET versions

This program permits composing and printing letters, flyers, advertisements, manuscripts, etc., using the COMMODORE PET and a printer.

Printing directives include line length, line spacing, left margin, centering and skip. Edit commands allow you to insert lines, delete lines, move lines and paragraphs, change strings, save files onto and load files from cassette (can be modified for disk), move up, move down, print and type.

Added features for the 16/32K version include string search for editing, keyboard entry during printing for letter salutations, justification, multiple printing and more.

A thirty page instruction manual is included.

The CmC Word Processor Program for the 8K PET is $29.50. The 16/32K version is $39.50.

Order direct or contact your local computer store.

VISA AND M/C ACCEPTED — SEND ACCOUNT NUMBER, EXPIRATION DATE AND SIGN ORDER.
ADD $1 PER ORDER FOR SHIPPING & HANDLING — FOREIGN ORDERS ADD 10% FOR AIR POSTAGE

CONNECTICUT microCOMPUTER, Inc.
150 POCONO ROAD
BROOKFIELD, CONNECTICUT 06804
TEL: (203) 775-9659 TWX: 710-456-0052

Graphics and the Challenger 1P

William L. Taylor
246 Flora Road N.W.
Leavittsburg, OH 44430

The Challenger computers have some interestings graphic capabilities. A discussion of the inner workings of the graphics and programs for using them are presented.

Introduction

Recently I purchased an OSI Challenger C1P, and I find its graphics and polled keyboard to be interesting tools for the programmer. But to the computer hobbyist with little experience in programming, it may seem very confusing. Since the C1P's introduction, I have seen few articles describing the graphics capabilities or use of the polled keyboard.

Part I

Programming the C1P in BASIC to utilize the graphics elements contained in the character generator and the polled keyboard are simple tasks when one understands how these functions work. This article will explain the polled keyboard functions and give a brief description of a program that I have written in Microsoft OSI BASIC to implement the graphics characters contained in the C1P character generator ROM.

The user of the C1P will find the keyboard a very interesting feature. Every key on the keyboard can be programmed and read under BASIC. This makes for real-time utilization of the keyboard. The program included in part I of this article shows how the keys are read with a PEEK statement and how the keyboard is strobed with a POKE statement. The keyboard is laid out in a matrice of eight rows and eight columns. To use the keyboard in a program, that is, a direct access in a running program; the programmer must first disable Control C. In the normal polling routine in a program the keyboard is interrogated to check for a Control C to signal the computer that a break is desired in the program. The Control C must be disabled.

To disable Control C, a flag in RAM must be set to 1. Normally the flag is set to 0. Next, the row that the key or keys that are to be read must be strobed. To do this, we POKE the row number. In the C1P, the rows are labeled R0 through R7. Each row has a decimal value assigned to it. The C1P keyboard is accessed in the following manner: POKE (57088), 127. This statement signals the keyboard that a row is to be examined for a key closure. To check the row for a closure the column in which the desired key is located must be examined. We do this with a PEEK statement, such as, IF PEEK(57088) = 127 THEN 100. This statement checks for the 1 key. If the 1 key were closed, then a jump to line 100 would be executed.

In the program that I have provided, you will see how the keyboard is polled to read the keys 1 through 8. If any of these keys are pressed the computer makes a decision concerning where to jump for a specific task. The following example shows how Control C is disabled and the row is strobed: 30 POKE 530,1: POKE (y),127. Variable Y is the keyboard location which is 57088 decimal. The next step is to read the columns in which the expected keys are located. For this we must PEEK the columns. This is done in lines 35 through 80 in the BASIC program. By examining the program further, we see that if a key from 1 to 8 is pressed, the program will jump to a subroutine. These subroutines are located at lines 100-800. It is in these subroutines that the actual plotting and writing of the graphics are accomplished.

At this point, a few words about the OSI C1P video display are in order. This display can produce up to four pages of alpha-numerics, which are in a 25 character line by 25 lines format. The alpha-numerics include upper case and lower case letters, the numeral set, punctuation marks, and 160 graphics elements.

Part I of this article is mostly concerned with the graphics elements and how they are executed in a BASIC program. To display any character on the

video monitor screen, the ASCII equivalent must be written in the video memory. This memory occupies 1 kilobyte of memory dedicated to the video display. This memory is located at D000 through D3FF hex, or 53379 to 54171 decimal. In the program I have set the video graphics pointer to point to mid-screen, as can be seen in the program at line 15. The mid-screen position is contained in the variable L. This is set to 53775 decimal.

The complete code set for the alpha-nimerics and the graphics elements is listed in the OSI "Graphics Manual" for the Challengers, so I will not delay in explaining all the elements or their codes, but rather, define the character that will be used in the enclosed program. In each of the subroutines in the BASIC program, the decimal code character is POKEd out to some video memory location. An example is 100 POKE L+A, 161. This places a square box on the screen depending on the value of L+A. If the program were just started and the 1 key were pressed and held down, the box would be placed at 53775 decimal, or mid-screen. If the key were kept held down the box would then be written at L+A again, but at 31 greater than the last box because A was incremented by 31 in the statement at line 110. As long as the 1 key is held down, the box would continue to be written at a location 31 places greater. This forms a diagonal downward to the left bottom of the screen. If the key is then released the program will halt and wait for another key to be pressed. If, for instance, the 6 key were next pressed, then the box would be written upward from the last point displayed on the screen where the diagonal ended. In examining the program, you will see that there are eight subroutines beginning at line 100 through line 850. These subroutines form a method for plotting the point where the box can be drawn from the use of the keys 1 through 8 on the keyboard. These keys are used as pointers, and they are defined in figure 1. The figure shows the direction of angle for each key. Each subroutine has a delay loop that allows the user to obtain a single point with a single key closure.

I have presented a brief description of the C1P's polled keyboard, and how to place a graphics element out to the video monitor screen with a BASIC program. This BASIC program allows an "etch-a sketch" type drawing on the monitor screen. From this quick description of the keyboard function and how a BASIC program can be used to read the keyboard in real-time, and from the explanation of how to place a graphics character out to the monitor screen with a BASIC program, you will be able to write similar programs using these techniques.

Listing 1

```
10 FOR R= 1 TO 32: PRINT: NEXT R
12 A=0:B=0:C=0:D=0
13 E=0:F=0:G=0:H=0
15 L=53775
20 Y=57088
30 POKE 530,1:POKE Y,127
35 IF PEEK(Y)=127 THEN 100
40 IF PEEK(Y)=191 THEN 200
45 IF PEEK(Y)=223 THEN 300
50 IF PEEK(Y)=239 THEN 400
55 IF PEEK(Y)=247 THEN 500
60 IF PEEK(Y)=251 THEN 600
65 IF PEEK(Y)=253 THEN 700
70 POKE Y,191
75 IF PEEK(Y)=127 THEN 800
80 GOTO 30
100 POKE L+A,161
110 A=A+31
140 FOR T= 1 TO 300:NEXT T
145 L=L+A
147 A=0
150 GOTO 30
200 POKE L+B,161
210 B=B+32
240 FOR T= 1 TO 300:NEXT T
245 L=L+B
247 B=0
250 GOTO 30
300 POKE L+C,161
310 C=C+33
340 FOR T= 1 TO 300:NEXT T
345 L=L+C
347 C=0
350 GOTO 30
400 POKE L+D,161
410 D=D+1
440 FOR T= 1 TO 300: NEXT T
445 L=L+D
447 D=0
450 GOTO 30
500 POKE L+E,161
510 E=E+-31
540 FOR T= 1 TO 300: NEXT T
545 L=L+E
547 E=0
550 GOTO 30
600 POKE L+F,161
610 F=F+ -32
640 FOR T= 1 TO 300: NEXT T
645 L=L+F
647 F=0
700 POKE L+G,161
710 G=G+ -33
740 FOR T= 1 TO 300: NEXT T
745 L=L+G
747 G=0
750 GOTO 30
800 POKE L+H,161
810 H=H+ -1
840 FOR T= 1 TO 300: NEXT T
845 L=L+H
847 H=0
850 GOTO 30
```

Part II

Now I will expand the basic programming principles pertaining to the development of graphics elements. This time we will develop graphic elements that represent large numbers as viewed on the system monitor screen. Please remember that the program following part 2 of this article is for demonstrating the methods of using a BASIC program to generate graphics elements utilizing the expanded graphic capabilities of the graphics generator that is resident in the C1P, and the OSI C2-4P computers.

I hope to give the reader the building blocks that will enable him to develop larger graphics programs using the techniques discussed here and in a companion article, in which I will give a BASIC program for a twelve hour clock that utilizes the large graphics numbers. The demonstration program is written in BASIC. It is written in subroutines and modular blocks. In the subroutines the graphic elements for the large numbers are generated and POKEd out to the C1P's video display. To begin, the subroutine at lines 1000 through 1100 will generate a large number (in this case, a large number 1).

To describe the operation of the subroutine, refer to the program listing 2. At line 1000 the screen parimeters are set up with a FOR -NEXT loop (FOR A = 5400 TO 54128 STEP 32). Line 1010 POKE A, 161: NEXT A. In these statement lines, the variable A will be incremented by 32 for every pass through the FOR-NEXT loop. When this portion of the subroutine is executed, the value 161 in statement line 1010 will place a white square block on the monitor screen beginning at the initial value in the A variable. In this instance the A variable will contain decimal 54000, located on the monitor screen near the bottom right hand corner. With every pass through the FOR-NEXT loop a white block will be placed 32 places ahead of the last video graphics character. On the C1P's monitor 32 places will place the next character directly below the last character placed on the screen. This FOR-NEXT loop in the subroutine will generate of place four white squares, one over the other,which will develop the graphics representation of the number one on the monitor screen.

Listing 2

```
1 REM NUMBER GRAPHICS DEMONSTRATOR
2 REM BY W.L.TAYLOR
3 REM JULY 4 1979
5 PRINT " THIS IS A DEMONSTRATION"
10 PRINT " OF THE C1P GRAPHICS AND LARGE NUMBERS"
20 PRINT " ALL NUMBERS FROM 1 TO 10 WILL BE DISPLAYED"
30 GOSUB 2900
39 REM INITIALIZE USR VECTOR FOR JUMP TO 2FE8
40 POKE 11,232: POKE 12,47
49 REM GENERATE RANDOM NUMBER FROM 0 TO 10
50 R= INT((11+1)*RND(1)-1)
52 REM COMPARE RANDOM NUMBER AND JUMP TO LARGE NUMBER TABLE
55 IF R > 11 THEN 50
56 IF R < Ø THEN 50
59 REM EXECUTE FAST SCREEN ERASE
60 X=USR(X)
65 IF R= 11 THEN GOSUB 1900
67 IF R= 11 THEN GOSUB 1000
70 IF R= 1 THEN GOSUB 1000
80 IF R= 2 THEN GOSUB 1100
90 IF R= 3 THEN GOSUB 1200
100 IF R= 4 THEN GOSUB 1300
110 IF R= 5 THEN GOSUB 1400
120 IF R= 6 THEN GOSUB 1500
130 IF R= 7 THEN GOSUB 1600
140 IF R= 8 THEN GOSUB 1700
150 IF R= 9 THEN GOSUB 1800
160 IF R= 10 THEN GOSUB 1900: GOSUB 2000
165 IF R= 0 THEN GOSUB 2000
170 FOR I= 1 TO 1000: NEXT I
180 X= USR(X)
190 GOTO 50
999 REM GENERATE LSD 1
1000 FOR A= 54000 TO 54128 STEP 32
1010 POKE A,161:NEXT A
1020 RETURN

1099 REM GENERATE LSD 2
1100 FOR A= 54000 TO 54002
1110 POKE A,161: NEXT A
1120 POKE 54034,161
1130 FOR A= 54064 TO 54066
1140 POKE A,161: NEXT A
1160 POKE 54096,161
1170 FOR A= 54128 TO 54130
1180 POKE A,161: NEXT A
1190 RETURN
1199 REM GENERATE LSD 3
1200 FOR A= 54000 TO 54002
1210 POKE A,161: NEXT A
1220  FOR A=  54064 TO 54066
1240 POKE A,161: NEXT A
1250 POKE 54098,161
1260 FOR A= 54128 TO 54130
1270 POKE A,161: NEXT A
1280 RETURN
1299 REM GENERATE LSD 4
```

At this point I will give a brief description of the BASIC program, explaining the unique features. This will give the user a better understanding of how the graphic characters can be utilized in other programs, such as games, clock programs, etc. In the BASIC program at line 30, a jump to subroutine at line2900 will load a machine language subroutine in user memory. that will be used for an ultra-fast screen erase when needed by the Main Line BASIC program. The Machine Language object code for the fast screen erase routine is stored in DATA statements at lines 3000 through 3030.

This data is read with a READ statement and POKEd into user memory at 12264 decimal through 12287 decimal. This corresponds with 2FE8 Hex through 2FFE Hex. The machine code routine when executed with the BASIC program will clear the last two pages of screen memory (that is, the bottom half of the C1p's monitor screen). This was done so that the user could utilize the top half for displaying a message and have it remain until the need to erase that half of the screen is desired. After the machine code is loaded into user memory, a RETURN from subroutine will be executed and the program will return to line 40, where the USR vector will be initialized to point to the beginning of the fast screen routine in user memory. The USR vector locations in the C1P are located at 11 and 12 decimal or OB and OC Hex. At line 50 a random number is generated and stored in the R variable. The statements at lines 55 and 56 insure that the random number will be only 0 through 10. The statement at line 60 will execute the fast screen erase. This is the USR function of BASIC, which causes a jump to the USR Vector at 11 and 12, where the jump to the fast screen erase is located. After the fast screen erase routine has been executed and the Op code Hex 60 is reached in the machine code routine, a return to BASIC will be executed and continue at line 65. The program forms line 65 through 165, is a table where the random number from the random number generator is compared to fixed constants. If the random number equals any of the constants, a jump to the subroutine that generates that number will occur. At line 170, the FOR-NEXT loop will allow the last generated video display to be viewed for the period of time that was set in the loop. The statement in line 180, calls up the fast screen erase machine code routine. The statement at line 190 forces a new pass through the mainline program.

From the program listing, you will see that the formation of the video graphics digits are developed in subroutines. These subroutines begin at

line 1000. There is a subroutine for each of the least significant digit and a subroutine for the next most digit. To develope the digit 10, we must use two of the subroutines. This would also be the case for any number greater than 10. The program is separated by REM statements. Each module will begin with a REM statement that defines the function of the subroutine, and if the reader analyses each module he will get a clear picture of how the numbers are generated and placed on the monitor screen.

The program listing beginning at line 3500, gives the object code listing for the fast screen erase. This is the machine code that is loaded into user memory when the BASIC program initializes the user memory through the BASIC subroutine at line 2899. The BASIC program listing has the fast screen erase routine loaded at 12264 to 12287 decimal. This was loaded at the top of a 12k memory. If your C1P does not have this much memory, you will have to change the program to work with the amount of memory that you may have in your system. The program listing gives the necessary changes for either an 8K or 4K memory system. These changes are listed starting at line 3500. A word of caution must be conveyed at this time. The user must set the memory size of his machine to reflect the size of memory that will allow the machine code routine to be intered and protected. That is, the memory size must be set when bringing up BASIC to less than the beginning of the machine code routine. If your system has only 4K of memory, set the memory size to 4050 decimal. If your memory has 8K, set the memory size to 8160. If you should have 12K, as my memory does, then set the size to 12263. Be sure that you change subroutine beginning at 2899 for your personal system depending on the amount of memory your system has available.

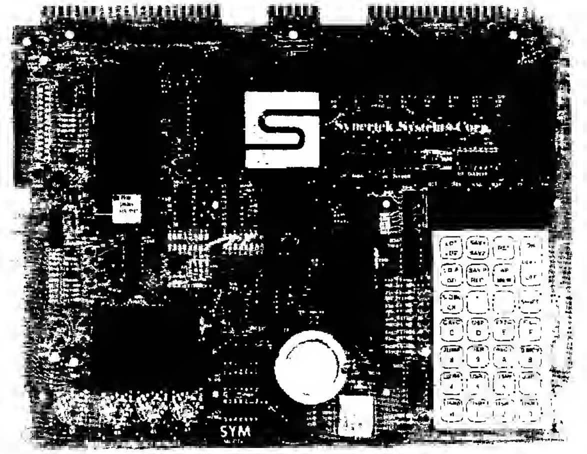
In conclusion, I have presented what I think will help you with the programming techniques needed to-understand the inner workings of the C1P's graphics capabilities, and the use of BASIC as a tool to be utilized with the graphics capabilities of the C1P, or other Challenger computers. The developement of large graphics numbers is only one example of how the expanded graphics set of the C1P can be used. The same techniques used in this article can be utilized for more complex exploration of the graphics and BASIC programming functions to develope programs such as games etc. In a future article, I will further expand the example program here to include a larger number set and have the C1P function as a twelve hour clock running under a BASIC program. Until then, good luck.

```
1300 FOR A= 54000 TO 54064 STEP 32
1310 POKE A,161: NEXT A
1320 FOR A= 54064 TO 54066
1330 POKE A,161: NEXT A
1340 FOR A= 54002 TO 54130 STEP 32
1350 POKE A,161: NEXT A
1360 RETURN
1399 REM GENERATE LSD 5
1400 FOR A= 54000 TO 54002
1410 POKE A,161: NEXT A
1420 FOR A= 54064 TO 54066
1425 POKE A,161: NEXT A
1430 FOR A= 54128 TO 54130
1440 POKE A,161: NEXT A
1450 POKE 54032,161: POKE 54098,161
1460 RETURN

1499  REM GENERATE LSD 6
1500 FOR A= 54000 TO 54002
1510 POKE A,161: NEXT A
1520 FOR A= 56064 TO 54066
1530 POKE A,161: NEXT A
1540 FOR A= 54128 to 54130
1550 POKE A,161:NEXT A
1560 POKE 54032,161: POKE 54096,161: POKE 54098,161
1570 RETURN
1599 REM GENERATE  LSD 7
1600 FOR A= 54000 TO 54002
1610 POKE A,161: NEXT A
1620 FOR A= 54002 TO 54130 STEP 32
1630 POKE A,161: NEXT A
1640 RETURN
1699 REM GENERATE LSD 8
1700 FOR A= 54000 TO 54128 STEP 32
1710 POKE A,161; NEXT A
1720 FOR A= 54002 TO 54130 STEP 32
1730 POKE A,161: NEXT A
1740 FOR A= 54001 TO 54129 STEP 64
1750 POKE A,161: NEXT A
1760 RETURN
1799 REM GENERATE LSD 9
1800 FOR A= 54002 TO 54130 STEP 32
1810 POKE A,161: NEXT A
1820 FOR A= 54000 TO 54002
1830 POKE A,161.: NEXT A
1840 FOR A= 54064 TO 54066
1850 POKE A,161: NEXT A
1860 FOR A= 54128 TO 54130
1870 POKE A,161: NEXT A
1880 POKE 54032,161
1890 RETURN
1899 REM GENERATE SMD 1
1900 FOR A= 53998 TO 54126 STEP 32
1910 POKE A,161: NEXT A
1930 RETURN
1999 REM GENERATE LSD 0
```

```
2000 FOR A= 54000 TO 54002
2010 POKE A,161: NEXT A
2020 FOR A= 54000 TO 54128 STEP 32
2030 POKE A,161: NEXT A
2040 FOR A= 54002 TO 54130 STEP 32
2050 POKE A,161: NEXT A
2060 POKE 54129,161
2070 RETURN
2899 REM FAST ERASE ROUTINE MACHINE CODE LOAD
2900 FOR R= 12264 TO 12287
2920 READ F: POKE R,F: NEXT R
2930 RETURN
3000 DATA 169,32,160,4,162,0,157,0
3010 DATA 210,232,208,250,238,240
3020 DATA 47,136,208,244,169,210
3030 DATA 141,240,47,96
3500 REM MACHINE CODE FAST SCREEN ERASE
3510 REM LOADS AT HEX  2FE8 TO  2FFF
3520 REM 2FE8  A9 20 A0 04 A2 00 9D 00 D2 E8 D0 FA
3530 REM EE F0 2F 88 D0 F4 A9 D2 8D F0 2F 60
3540 REM TYPE CONTROL C TO END
3550 REM CHANGE LINE 2900 TO (FOR R= 4072 TO 4095) FOR A 4K
     SYSTEM
3560 REM CHANGE LINE 3000 TO 3030 TO REFLECT THE NEXT LIST
              DATA 169,32,160,4,162,0,157,0
         3010 DATA 210,232,208,250,238,240
         3020 DATA 15,136,208,244,169,210
         3030 DATA 141,240,15,96
3580 REM THESE ARE FOR A 4K C1P
3590 REM CHANGE LINE 40 (40 POKE 11,232: POKE 12,15)
```

SYM-1, 6502-BASED MICROCOMPUTER

- FULLY-ASSEMBLED AND COMPLETELY INTEGRATED SYSTEM that's ready-to-use
- ALL LSI IC'S ARE IN SOCKETS
- 28 DOUBLE-FUNCTION KEYPAD INCLUDING UP TO 24 "SPECIAL" FUNCTIONS
- EASY-TO-VIEW 6-DIGIT HEX LED DISPLAY
- KIM-1* HARDWARE COMPATIBILITY
  The powerful 6502 8-Bit MICROPROCESSOR whose advanced architectural features have made it one of the largest selling "micros" on the market today.
- THREE ON-BOARD PROGRAMMABLE INTERVAL TIMERS available to the user, expandable to five on-board.
- 4K BYTE ROM RESIDENT MONITOR and Operating Programs.
- Single 5 Volt power supply is all that is required.
- 1K BYTES OF 2114 STATIC RAM onboard with sockets provided for immediate expansion to 4K bytes onboard, with total memory expansion to 65, 536 bytes.
- USER PROM/ROM: The system is equipped with 3 PROM/ROM expansion sockets for 2316/2332 ROMs or 2716 EPROMs
- ENHANCED SOFTWARE with simplified user interface
- STANDARD INTERFACES INCLUDE:
  —Audio Cassette Recorder Interface with Remote Control (Two modes: 135 Baud KIM-1* compatible, Hi-Speed 1500 Baud)
  —Full duplex 20mA Teletype Interface
  —System Expansion Bus Interface
  —TV Controller Board Interface
  —CRT Compatible Interface (RS-232)
- APPLICATION PORT: 15 Bi-directional TTL Lines for user applications with expansion capability for added lines
- EXPANSION PORT FOR ADD-ON MODULES (51 I/O Lines included in the basic system)
- SEPARATE POWER SUPPLY connector for easy disconnect of the d-c power
- AUDIBLE RESPONSE KEYPAD



Synertek has enhanced KIM-1* software as well as the hardware. The software has simplified the user interface. The basic SYM-1 system is programmed in machine language. Monitor status is easily accessible, and the monitor gives the keypad user the same full functional capability of the TTY user. The SYM-1 has everything the KIM-1* has to offer, plus so much more that we cannot begin to tell you here. So, if you want to know more, the SYM-1 User Manual is available, separately.

SYM-1 Complete w/manuals	**$269.00**
SYM-1 User Manual Only	**7.00**
SYM-1 Expansion Kit	**75.00**

Expansion includes 3K of 2114 RAM chips and 1-6522 I/O chip.

SYM-1 Manuals: The well organized documentation package is complete and easy-to-understand.

SYM-1 CAN GROW AS YOU GROW. Its the system to BUILD-ON. Expansion features that are soon to be offered:

***BAS-1 8K Basic ROM (Microsoft)**	**$159.00**
***KTM-2 TV Interface Board**	**349.00**

We do honor Synertek discount coupons

QUALITY EXPANSION BOARDS DESIGNED SPECIFICALLY FOR KIM-1, SYM-1 & AIM 65

These boards are set up for use with a regulated power supply such as the one below, but, provisions have been made so that you can add onboard regulators for use with an unregulated power supply. But, because of unreliability, we do not recommend the use of onboard regulators. All I.C.'s are socketed for ease of maintenance. *All boards carry full 90-day warranty.*

All products that we manufacture are designed to meet or exceed industrial standards. All components are first qualtiy and meet full manufacturer's specifications. All this and an extended burn-in is done to reduce the normal percentage of field failures by up to 75%. To you, this means the chance of inconvenience and lost time due to a failure is very rare; but, if it should happen, we guarantee a turn-around time of less than forty-eight hours for repair.

Our money back guarantee: If, for any reason you wish to return any board that you have purchased directly from us within ten (10) days after receipt, complete, in original condition, and in original shipping carton; we will give you a complete credit or refund less a $10.00 restocking charge per board.

VAK-1 8-SLOT MOTHERBOARD

This motherboard uses the KIM-4* bus structure. It provides eight (8) expansion board sockets with rigid card cage. Separate jacks for audio cassette, TTY and power supply are provided. Fully buffered bus.

VAK-1 Motherboard	**$129.00**

VAK-2/4 16K STATIC RAM BOARD

This board using 2114 RAMs is configured in two (2) separately addressable 8K blocks with individual write-protect switches.

VAK-2 16K RAM Board with only 8K of RAM (½ populated)	**$239.00**
VAK-3 Complete set of chips to expand above board to 16K	**$175.00**
VAK-4 Fully populated 16K RAM	**$379.00**

VAK-5 2708 EPROM PROGRAMMER

This board requires a +5 VDC and ±12 VDC, but has a DC to DC multiplyer so there is no need for an additional power supply. All software is resident in on-board ROM, and has a zero-insertion socket.

VAK-5 2708 EPROM Programmer	**$269.00**

VAK-6 EPROM BOARD

This board will hold 8K of 2708 or 2758, or 16K of 2716 or 2516 EPROMs. *EPROMs not included.*

VAK-6 EPROM Board	**$129.00**

VAK-7 COMPLETE FLOPPY-DISK SYSTEM (May '79)

VAK-8 PROTYPING BOARD

This board allows you to create your own interfaces to plug into the motherboard. Etched circuitry is provided for regulators, address and data bus drivers; with a large area for either wire-wrapped or soldered IC circuitry.

VAK-8 Protyping Board	**$49.00**

POWER SUPPLIES

ALL POWER SUPPLIES are totally enclosed with grounded enclosures for safety, AC power cord, and carry a full 2-year warranty.

FULL SYSTEM POWER SUPPLY

This power supply will handle a microcomputer and up to 65K of our VAK-4 RAM. ADDITIONAL FEATURES ARE: Over voltage Protection on 5 volts, fused, AC on/off switch. Equivalent to units selling for $225.00 or more.

Provides +5 VDC @ 10 Amps & ±12 VDC @ 1 Amp

VAK-EPS Power Supply	**$125.00**

KIM-1* Custom P.S. provides 5 VDC @ 1.2 Amps and +12 VDC @ .1 Amps

KCP-1 Power Supply	**$41.50**

SYM-1 Custom P.S. provides 5 VDC @ 1.4 Amps

VCP-1 Power Supply	**$41.50**

*KIM is a product of MOS Technology

RNB ENTERPRISES INCORPORATED

2967 W Fairmount Avenue
Phoenix AZ 85017
(602)265-7564

master charge VISA

Time of Day Clock and Calendar for the SYM-1

Casmir J. Suchyta, III
and Paul W. Zitzewitz
Univ. of Michigan, Dearborn
4901 Evergreen Road
Dearborn, MI 48128

Now you can have a Clock and Calander running in your SYM at the same time you are running programs in BASIC. The concepts presented can be easily generalized into other 'multi-task' operations.

Here is a machine language subroutine for the SYM-1 BASIC which keeps track of time and date while allowing BASIC programs to be run.

A useful adjunct to a microcomputer, especially one used in a system, is a continuously running clock which can be used to record the time at which events occur or to generate signals at specified times. The SYM-1 includes timers on the 6522 VIA chips which make implementation of such a clock easy. The clock can be started, set, and read from BASIC.

The clock is based on the use of the 6522 to generate a train of accurately spaced interrupts. The April, 1979, issue of MICRO contained an article by John Gieryic (page 31) which presented the techniques of setting up and servicing the interrupts. The clock is an adaptation of those techniques. The program consists of sections which set the clock, initialize the interrupt, service the interrupt, and update the clock. The clock-calendar needs to be reset only on February 29!

The program is loaded into the highest bytes of available memory. On a 4K machine this is $0F54-$0FFF. After the program is loaded, BASIC is initialized with Memory Size set at 3920 to avoid overwriting the program. The clock is set and started by the command PRINT USR(3924,M,d,h,m), where the four parameters represent the month, date, hour, and minute, respectively. The program stores the times, then initializes the interrupt and starts the timer as described in MICRO 11:31. The timer located at $ACxx was used to avoid interference with the cassette tape routines. Once every 1/20 second an interrupt occurs which is serviced in the routines starting at $0F90. Accumulator and registers are pushed on to the stack, then the 1/20 of seconds, seconds, minutes, and hours are incremented as needed. These four updates are done in an indexed loop, using a table of comparison values (20 fractions, 60 seconds, 60 minutes, 24 hours) stored at $0FE9 to see if the next timing unit should be incremented. The days and months cannot be incremented in the same loop, and so are done in the routines starting at $0FBD. There is a comparison table giving the number of days (plus one) in each month starting at $0FF4 used to determine if the month should be incremented. When all needed increments are made the flag is cleared and the saved registers pulled back from the stack.

The clock may be read from BASIC by PEEKing at the appropriate storage locations. To print the date and time in the form 7/20/1979 17:45:02 execute the commandPRINTPEEK(4083)"/" PEEK(4082)''/1979 "PEEK(4081)":"PEEK(4080)":"PEEK-(4079). The number of the month in the date can be replaced by a three letter abbreviation by using the following short program to print the date.

```
1 A$ = "JANFEBMARAPRMAYJUN-
JULAUGSEPOCTNOVDEC"
2 MO = 1 + 3*(PEEK(4083) - 1)
3 PRINT
MID$(A$,MO,3);PEEK)4082);",1979"
```

Starting each program with this routine will let you know exactly when you did each job. Another use of the clock is to serve as an alarm clock. You may want the SYM to turn on a light, or start an experiment at a certain time. To do this include a tight loop which includes an IF statement comparing one or more of the storage locations with the desired time. When the comparison is good, the loop will be exited and the computer can execute the command.

```
.V F54-FFF
0F54 3C FE 0F 68 8D F1 0F 68,E3
0F5C 68 8D F2 0F 68 68 8D F3,2F
0F64 0F 68 20 86 8B A9 90 8D,9C
0F6C 7E A6 A9 0F 8D 7F A6 A9,D3
0F74 C0 8D 0E AC AD 0D AC 29,69
0F7C EF 8D 0D AC A9 C0 8D 0B,6F
0F84 AC A9 50 8D 06 AC A9 C3,BF
0F8C 8D 05 AC 60 08 48 8A 48,7F
0F94 98 48 D8 A0 00 A9 00 99,19
0F9C ED 0F C8 C0 05 F0 1A 18,C4
0FA4 B9 ED 0F 69 01 D9 E8 0F,B3
0FAC F0 EB 99 ED 0F A9 C3 8D,1C
0FB4 07 AC 68 A8 68 AA 68 28,81
0FBC 40 18 AD F2 0F 69 01 AE,9F
0FC4 F3 0F DD F3 0F F0 06 8D,03
0FCC F2 0F 4C B1 0F A9 01 8D,47
0FD4 F2 0F E8 E0 0D F0 06 8E,A1
0FDC F3 0F 4C B1 0F A2 01 8E,E0
0FE4 F3 0F 4C B1 0F 14 3C 3C,7A
0FEC 18 0C 05 1E 34 0E 15 05,11
0FF4 20 1D 20 1F 20 1F 20 20,CC
0FFC 1F 20 1F 20,8A
 493A
```

Listing: Time-of-Day Clock and Calendar

```
                 ORG        $0F54
                 MIN      * $0FF0
                 HR       * $0FF1
                 DAY      * $0FF2
                 MON      * $0FF3
                 COMP     * $0FED
                 ACCESS   * $8B86
0F54  8C F0 0F  Setime      STY   MIN     Stores minutes
0F57  68                    PLA           Pulls hours
0F58  8D F1 0F              STA   HR       and stores
0F5B  68                    PLA           Pulls Day
0F5C  68                    PLA            and
0F5D  8D F2 0F              STA   DAY       stores
0F60  68                    PLA           Pulls month
0F61  68                    PLA            and
0F62  8D F3 0F              STA   MON       stores
0F65  68                    PLA           Clears stack
0F66  20 86 8B              JSR   ACCESS  Unwrite protect the system RAM
0F69  A9 90                 LDAim $90     Store low
0F6B  8D 7E A6              STA   $A67E    byte IRQ
0F6E  A9 0F                 LDAim $0F     Store high
0F70  8D 7F A6              STA   $A67F    byte IRQ
0F73  A9 C0                 LDAim $C0     Set
0F75  8D 0E AC              STA   $AC0E    IER
0F78  AD 0D AC              LDA   $AC0D   Set
0F7B  29 BF                 AND   $BF
0F7D  8D 0D AC              STA   $AC0D    IFR
0F80  A9 C0                 LDAim $C0     Set
0F82  8D 0B AC              STA   $AC0B    ACR
0F85  A9 50                 LDAim $50     Set
0F87  8D 06 AC              STA   $AC06    and
0F8A  A9 C3                 LDAim $C3       start
0F8C  8D 05 AC              STA   $AC05      timer
0F8F  60                    RTS           return
0F90  08        Intrpt      PHP           Push processor
0F91  48                    PHA                Accum
0F92  8A                    TXA
0F93  48                    PHA                X reg
0F94  98                    TYA
0F95  48                    PHA                Y reg
0F96  D8        INCR        CLD           Clear dec flag
0F97  A0 00                 LDYim $00     Zero Y
0F99  A9 00      LOOP       LDAim $00          A
0F9B  99 ED 0F              STAy  COMP    Zeros counter
0F9E  C8                    INY           To next counter
0F9F  C0 05                 CPYim $05     Need new day?
0FA1  F0 1A                 BEQ   ADDAY   Go to it
0FA3  18                    CLC           Clear carry
0FA4  B9 ED 0F              LDAy  COMP    Get counter value
0FA7  69 01                 ADCim $01     increment
0FA9  D9 E8 0F              CMPy  HIGH-1  Comp with highest
0FAC  F0 EB                 BEQ   LOOP    Go to zero and carry to next
0FAE  99 ED 0F              STAy  COMP    Store new value
0FB1  A9 C3     RETN        LDAim $C3     Finished: clear
0FB3  8D 07 AC              STA   $AC07    interrupt flag
0FB6  68                    PLA           Restore
0FB7  A8                    TAY            Y reg
0FB8  68                    PLA
0FB9  AA                    TAX            X reg
0FBA  68                    PLA            Accum
0FBB  28                    PLP            Processor
0FBC  40                    RTI           Leave
0FBD  18        ADDAY       CLC           Clear carry
0FBE  AD F2 0F              LDA   DAY     Get day
0FC1  69 01                 ADCim $01     increment
0FC3  AE F3 0F              LDX   MON     Put month in x reg
0FC6  DD F3 0F              CMPx  MON     See if at last day
0FC9  F0 06                 BEQ   REDAY   Yes, go to month change
0FCB  8D F2 0F              STA   DAY     Save new day
0FCE  4C B1 0F              JMP   RETN    Leave
0FD1  A9 01     REDAY       LDAim $01     Back to day one!
0FD3  8D F2 0F              STA   DAY     Save
0FD6  E8                    INX           To next month
0FD7  E0 0D                 CPX   $0D     At end of year (13)?
0FD9  F0 06                 BEQ   END     Go to reset year
0FDB  8E F3 0F              STX   MON     Save new month
0FDE  4C B1 0F              JMP   RETN    Leave
0FE1  A2 01     END         LDXim $01     Back to January (1)
0FE3  8E F3 0F              STX   MON     Save
0FE6  4C B1 0F              JMP   RETN    Leave

0FE9  14 3C 3C  HIGH                      Table of highest values of
0FEC  18 00 00                            fractions, seconds, minutes, hours,(dummy)
0FEF  00 00 00                            followed by storage area for fractions,
0FF2  00 00                               seconds, minutes, hours, days, months
0FF4  20 1D 20                            Table of max days in each month
0FF7  1F 20 1F                            (plus one) for the twelve months.
0FFA  20 20 1F
0FFD  20 1F 20
```

APPLE II Speed Typing Test With Input Time Clock

John Broderick, CPA
8635 Shagrock
Dallas, TX 75238

So, you think you are a pretty fast typist! Care to take a Speed Typing Test on your APPLE?

The qiuck brwn fpx jumped ovre ...

The speed typing test is a must for all APPLEliers, like myself, who consider themselves expert typists. However, I did not set out to write a typing test, but to make an input subroutine (GOSUB 8400) which puts the user under the pressure of a time clock.

Try the program below:

```
2000 call-936:

2010 VV = 10: rem set VTAB

2020 TT =  1: rem set TAB

2030 GOSUB 8400

2040 GOTO 2000
```

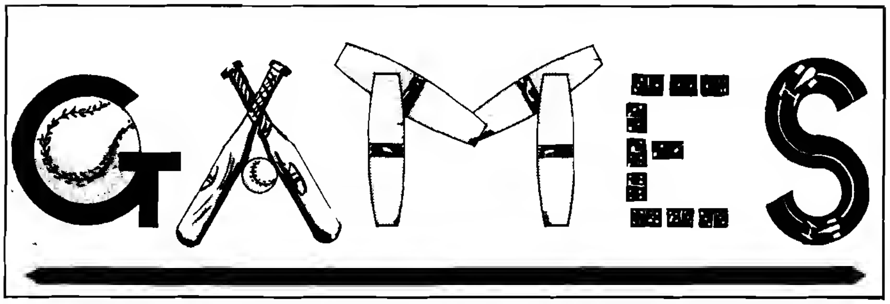
You should hear and see the time at the bottom of the screen with the seconds and tenths of seconds flying by as you type in an alpha-numeric string.

Subroutine 8400 reads the keyboard in line 8434 with K equal to the ASCII number. Line 8447 subtracts 159 from ASCII so that now K is equal to the position of the equivalent character in string A$ (line 8406). So you can see that we are slowly building up two words in W$ at line 8447 by adding, to the end of string W$, the next letter coming in on the keyboard until the ASCII equivalent of carriage return (141) is detected at line 8444.

Now when the princess falls into the snake pit, if she doesn't make the right decision fast enough the snakes will probably get her.

```
        WRITTEN BY JOHN BRODERICK
            DALLAS, TEXAS
 14 REM     JUNE 21, 1979
        SUBROUTINE 8400 IS A SELF
        CONTAINED INPUT TIME CLOCK

 16 REM DEFINE VV=VTAB & TT=TAB
        THEN GOSUB8400-THIS DOES THE
        SAME AS AN ORDINARY INPUT W$

 20 REM COPYWRITED-CAN NOT BE SOLD
        BUT CAN BE GIVEN AWAY
 40 DIM TYPE$(250): CALL -936: POKE
    33,36
 80 INPUT "DO YOU WISH TO MAKE UP YO
    UR OWN TEST SENTENCE  Y/N ? "
    ,TYPE$
 84 IF TYPE$#"Y" THEN 90: PRINT
    : PRINT "ENTER TEST SENTENCE NOW
    ": PRINT : PRINT : INPUT TYPE$
    : GOTO 100
 90 TYPE$="NOW IS THE TIME FOR ALL G
    OOD MEN TO COME TO THE AID OF TH
    EIR COUNTRY."
100 CALL -936: PRINT :ERR=0: PRINT
    "YOU ARE TAKING A SPEED TYPING T
    EST"
120 PRINT : PRINT "TYPE THE NEXT SEN
    TENCE APPEARING ON  THE SCREEN A
    S FAST AS YOU CAN"
130 FOR I=1 TO 4000: NEXT I: REM

135 REM --- BODY OF PROGRAM ----
140 CALL -936:ERR=0
150 VV=13: REM   SET SUBROUT VTAB
160 TT=1: REM  SET SUBROUT  TAB
170 VTAB (9): TAB 1: PRINT TYPE$
    : GOSUB 8400
180 VTAB (16): TAB 1
200 IF W$=TYPE$ THEN 510: REM

204 REM   COMPUTE ERRORS 210-410
210 FOR I= LEN(W$) TO LEN(TYPE$
    ):W$(I+1)=B$(1,1): NEXT I
220 FOR I=1 TO LEN(TYPE$): IF I>
     LEN(W$) THEN ERR=ERR+1: IF
    I>LEN(W$) THEN NEXT I
230 IF W$(I,I)#TYPE$(I,I) THEN
    ERR=ERR+1: NEXT I
400 PRINT : PRINT : CALL -198: PRINT
    "   ";ERR;" ERRORS      HIT RETU
    RN": GOTO 520
410 CALL -198: PRINT "   ";ERR;" ERRO
    RS";"    HIT RETURN"
500 REM - COMPUTE  WPM
501 T=(X*23)+J:L= LEN(TYPE$): IF
    L<1 THEN 520
502 L=L-(ERR*6): IF L<2 THEN GOTO
    506
503 WPM=(L*12*20)/T
506 VTAB (24): TAB 30: PRINT WPM;
    " WPM": VTAB (16): TAB 1: RETURN

510 PRINT " CORRECT  -  HIT RETURN"
    : PRINT : PRINT : PRINT :
520 GOSUB 500: INPUT W$:WPM=0: GOTO
    140: REM


8400 REM -SUBROUTINE TO INPUT VIA
          KEYBOARD TO RETAIN AND
          INPUT WORD IN W$
8405 IF SWITCH=1 THEN 8407:SWITCH=
     1: DIM W$(255),A$(70),B$(2)
     :B$="  "
8406 A$=" #$%&'()*+,-./0123456789:;
     '=?@ABCDEFGHIJKLMNOPQRSTUVWXYZ
     /m  "
8407 Y=T: POKE -16336,0:W$="  ":
     X=0:J=0
8410 FOR U=1 TO 250
8412 REM  USER AREA HERE X=SECONDS
          SO USER CAN TEST X LIKE
          IF X=12 THEN RETURN
8430 J=J+1: IF J<23 THEN 8434:X=
     X+1:J=0
8431 FOR BB=1 TO 3:KK= PEEK (-16336
     )- PEEK (-16336): NEXT BB: GOTO
     8434
8434 VTAB (24): TAB 13:U=U-1: PRINT
     X;".";J*10/23;"  SECONDS";:
     K= PEEK (-16384)
8437 IF K#136 THEN 8444:Y=Y-1
8438 VTA8 (VV): TAB TT+Y-1: PRINT
     B$(1,1)
8440 W$(1)=W$(1, LEN(W$)-1)
8441 VTAB (13): TAB 1: PRINT W$
8442 POKE -16368,0: NEXT U
8444 IF K=141 THEN 8540: IF K<160
      THEN NEXT U
8447 K=K-159:W$(Y)=A$(K,K)
8461 POKE -16368,0: VTAB (VV): TAB
     TT: PRINT W$:Y=Y+1: NEXT U
8540 Y=1: CALL -756: RETURN
```

Instant Software, the Best Value



We have it all—conflict simulation, games of chance, fast-paced fun. We have a package of good times for every taste.

TREK-X Command the Enterprise as you scour the quadrant for enemy warships. This package not only has superb graphics, but also includes programming for optional sound effects. A one-player game for the PET 8K. **Order No. 0032P $7.95.**

GOLF Without leaving the comfort of your chair, you can enjoy a computerized 18 holes of golf with a complete choice of clubs and shooting angles. You need never cancel this game because of rain. One or two players can enjoy this game on the Apple with Applesoft II and 20K. **Order No. 0018A $7.95.**

BOWLING/TRILOGY Enjoy two of America's favorite games transformed into programs for your Apple:
•**Bowling**—Up to four players can bowl while the Apple sets up the pins and keeps score. Requires Applesoft II.
•**Trilogy**—This program can be anything from a simple game of tic-tac-toe to an exercise in deductive logic. For one player.
This fun-filled package requires an Apple with 20K. **Order No. 0040A $7.95.**

TANGLE/SUPERTRAP These two programs require fast reflexes and a good eye for angles:
•**Tangle**—Make your opponent crash his line into an obstacle.
•**Supertrap**—This program is an advanced version of Tangle with many user control options.
Enjoy these exciting and graphically beautiful programs. For one or two players with an 8K PET. **Order No. 0029P $7.95.**

CHECKERS/BACCARAT Play two old favorites with your PET.
•**Checkers**—Let your PET be your ever-ready opponent in this computer-based checkers program.
•**Baccarat**—You have both Casino- and Blackjack-style games in this realistic program.
Your PET with 8K will offer challenging play anytime you want. **Order No. 0022P $7.95.**

CASINO I These two programs are so good, you can use them to check out and debug your own gambling system!
•**Roulette**—Pick your number and place your bet with the computer version of this casino game. For one player.
•**Blackjack**—Try out this version of the popular card game before you go out and risk your money on your own "surefire" system. For one player.
This package requires a PET with 8K. **Order No. 0014P $7.95.**

CASINO II This craps program is so good, it's the next best thing to being in Las Vegas or Atlantic City. It will not only play the game with you, but will also teach you how to play the odds and make the best bets. A one-player game, it requires a PET 8K. **Order No. 0015P $7.95.**

TURF AND TARGET Whether on the field or in the air, you'll have fun with the Turf and Target package. Included are:
•**Quarterback**—You're the quarterback as you try to get the pigskin over the goal line. You can pass, punt, hand off, and see the result of your play with the PET's superb graphics.
•**Soccer II**—Play the fast-action game of soccer with four playing options. The computer can play itself or a single player; two can play with computer assistance, or two can play without help.
•**Shoot**—You're the hunter as you try to shoot the bird out of the air. The PET will keep score.
•**Target**—Use the numeric keypad to shoot your puck into the home position as fast as you can. To run and score, all you'll need is a PET with 8K. **Order No. 0097P $7.95.**

DUNGEON OF DEATH Battle evil demons, cast magic spells, and accumulate great wealth as you search for the Holy Grail. You'll have to descend into the Dungeon of Death and grope through the suffocating darkness. If you survive, glory and treasure are yours. For the PET 8K. **Order No. 0064P $7.95.**

PET DEMO I You can give yourself, your family, and your friends hours of fun and excitement with this gem of a package.
•**Slot Machine**—You won't be able to resist the enticing messages from this computerized one-armed bandit.
•**Chase**—You must find the black piece as you search through the ever-changing maze.
•**Flying Pheasant**—Try to shoot the flying pheasant on the wind.
•**Sitting Ducks**—Try to get your archer to shoot as many ducks as possible for a high score.
•**Craps**—It's Snake Eyes, Little Joe, or Boxcars as you roll the dice and try to make your point.
•**Gran Prix 2001**—Drivers with experience ranging from novice to professional will enjoy this multi-leveled race game.
•**Fox and Hounds**—It's you against the computer as your four hounds try to capture the computer's fox.
For true excitement, you'll need a PET 8K. **Order No. 0035P $7.95.**

PENNY ARCADE Enjoy this fun-filled package that's as much fun as a real penny arcade—at a fraction of the cost!
•**Poetry**—Compose free verse poetry on your computer.
•**Trap**—Control two moving lines at once and test your coordination.
•**Poker**—Play five-card draw poker and let your PET deal and keep score.
•**Solitaire**—Don't bother to deal, let your PET handle the cards in this "old favorite" card game.
•**Eat-Em-Ups**—Find out how many stars your Gobbler can eat up before the game is over.
These six programs require the PET with 8K. **Order No. 0044P $7.95.**

MIMIC Test your memory and reflexes with the five different versions of this game. You must match the sequence and location of signals displayed by your PET. This one-player program includes optional sound effects with the PET 8K. **Order No. 0039P $7.95.**

MIMIC (see description for the PET version 0039P) This package requires the Apple 24K. **Order No. 0025A $7.95.**

ARCADE I This package combines an exciting outdoor sport with one of America's most popular indoor sports:
•**Kite Fight**—It's a national sport in India. After you and a friend have spent several hours maneuvering your kites across the screen of your PET, you'll know why!
•**Pinball**—By far the finest use of the PET's exceptional graphics capabilities we've ever seen, and a heck of a lot of fun to boot.
Requires an 8K PET. **Order No. 0074P $7.95.**

ARCADE II One challenging memory game and two fast-paced action games make this one package the whole family will enjoy for some time to come. Package includes:
•**UFO**—Catch the elusive UFO before it hits the ground!
•**Hit**—Better than a skeet shoot. The target remains stationary, but you're moving all over the place.
•**Blockade**—A two-player game that combines strategy and fast reflexes.
Requires 8K PET. **Order No. 0045P $7.95.**

Business

On the bottom line you'll know that our business packages mean better business for you.

ACCOUNTING ASSISTANT This package will help any businessman solve many of those day-to-day financial problems. Included are:
•**Loan Amortization Schedule**—This program will give you a complete breakdown of any loan or investment. All you do is enter the principal amount, interest rate, term of the loan or investment, and the number of payments per year. You see a month-by-month list of the principal, interest, total amount paid, and the remaining balance.
•**Depreciation Schedule**—You can get a depreciation schedule using any one of the following methods: straight line, sum of years-digits, declining balance, units of production, or machine hours. Your computer will display a list of the item's lifespan, the annual depreciation, the accumulated depreciation, and the remaining book value. This package requires the PET 8K. **Order No. 0048P $7.95.**

ACCOUNTING ASSISTANT (see the description for the PET version 0048P) This package requires the Apple 16K. **Order No. 0088A $7.95.**

SUMTEST: A Memory Test Routine for the 6502

S. Felton Mitchell, Jr.
c/o The Bit Stop
P.O. Box 973
Mobile, AL 36601

No microcomputer is better than its RAM memory. Here is a RAM memory test that can be adapted to any 6502 based system.

SUMTEST is a short (107 byte) machine language program to test memory. The algorithm is not original with me, as I have seen similar routines published for the 8008, 8080, and 6800 microprocessors. I have not, however, seen the SUMTEST algorithm used in a 6502 memory test routine.

SUMTEST will detect all "stuck" bits, and will print the error address and the offending bit pattern. SUMTEST will also detect address sensitive errors, such as the sct of writing to hex location 0208 changing the contents of hex location 03BC. The sensitive address errors can result from shortened address lines or interaction of adjacent memory cells within a memory chip. SUMTEST will not detect byte sensitive memory failures (except by accident).

The routine is assembled to reside in the first part of page 01, the stack page for the 6502. The stack page is intentionally used due to the fact that if your 6502 machine is running, at least the few bytes of page 01 used by the stack are "good." The routine can be relocated elsewhere in memory if you want to test the first part of page 01 where the routine resides. You will not be able to test the top few bytes of page 01 used as stack space by theprogram, as any modification of the stack area while the routine is running will result in a program bomb.

The program as currently assembled uses KIM output routines. If your machine is not a KIM (as mine is not), then you will have to substitute your system print routines. The print routines are defined at the beginning of the listing supplied.

The algorithm used calculates a data byte to store each memory location

```
SYMBOL TABLE 3ØØØ 3Ø96
BGNADH ØØ81   BGNADL ØØ8Ø   CMPADL Ø161   COUNTR ØØ84
CRLF   1E2F   ENDADH ØØ83   ENDADL ØØ82   ERROR  Ø12B
INCPTR Ø15B   INIT   Ø1ØØ   LOOPA  Ø1Ø8   LOOPB  Ø115
ONCE   Ø121   OUTCH  1EAØ   OUTSP  1E9E   PRTBYT 1E3B
RETURN Ø11C   RTN    Ø16B   SETEM  Ø14A   SUMTST Ø1ØØ
SUMUM  Ø153   TEST   Ø1Ø3   TMPADH ØØ86   TMPADL ØØ85
TMPY   ØØ87
```

Figure 1

```
SYMBOL TABLE 3ØØØ 3Ø96
BGNADL ØØ8Ø   BGNADH ØØ81   ENDADL ØØ82   ENDADH ØØ83
COUNTR ØØ84   TMPADL ØØ85   TMPADH ØØ86   TMPY   ØØ87
INIT   Ø1ØØ   SUMTST Ø1ØØ   TEST   Ø1Ø3   LOOPA  Ø1Ø8
LOOPB  Ø115   RETURN Ø11C   ONCE   Ø121   ERROR  Ø12B
SETEM  Ø14A   SUMUM  Ø153   INCPTR Ø15B   CMPADL Ø161
RTN    Ø16B   CRLF   1E2F   PRTBYT 1E3B   OUTSP  1E9E
OUTCH  1EAØ
```

Figure 2

to be tested by adding the high order address and the low address of each location to a "counter" byte. After all locations to be tested have been filled with their calculated data byte, the routine then recalculates the data byte that should be stored in each location and checks it against the actual contents of the location. If the data in memory is different from the calculated value, then the location and offending bit pattern are printed. As previously mentioned, there can be differences due to "stuck" bits or interaction of memory locations. Each time that the routine is successfully executed, it will print a "plus" on the system terminal. To completely test the memory (adding all 256 possible "counter" byte combinations to the address), it is necessary to have 256 "plusses" printed on your terminal. The program listing is exhaustively commented and should be pretty much self expanitory for even a novice machine language programmer.

To test 4K of memory occupying hex locations 200 to 2FFF, enter 00 at 0080, 20 at 0081, 00 at 0082, and 30 at 0083 (end address plus 1) and run at 0010. If no errors are detected, you will get a string of plusses on your terminal. Remember that 256 plusses are required to complete the test. An example of an error would be a carriage return line feed on the terminal, a four digit address (in hex), a space and a two digit number. The two digit number represents the bad bit pattern. Now convert the "bad bit" pattern to its binary equivalent. Each "1" in the binary pattern represents a bad bit at the memory location printed. If 23A840 was printed on your terminal, it would mean that bit 6 was bad at location 23A8. By reference to the memory board documentation, you should be able to determine which chip on the board is faulty.

An interesting observation was made during the developement of the program. My machine is a homebrew S100 bus, dual processor system. I have a 6502 and a 6800 on an S100 prototype board, each sharing all of the system except for a little PROM which is unique for each microprocessor. The system clock is derived from the clock generator in the 6502 (1MHz.). An equivalent SUMTEST program for the 6800 would cycle through my 24K of memory with no errors detected. The 6502 SUMTEST program would consistantly catch several bad bytes. Apparently there is a few nanosecond's difference in the timing of the two microprocessors, and that was just enough for some of the memory to fail. All of the memory that tested bad on the 6502 was purchased from one vendor as 450 nanosecond memory. So be aware that a few nanoseconds can make a big difference, and purchase your memory from a reputable supplier.

Listing 1

```
0100               SUMTST ORG   $0100  ASSEMBLE IN STACK PAGE
                    REMEMBER THAT THE ROUTINE DESTROYS THE CONTENTS OF
                    THE MEMORY TESTED.
0100               BGNADL .     $0080  START ADDRESS OF MEMORY TO BE
                                            TESTED
0100               BGNADH .     $0081
0100               ENDADL .     $0082  END ADDRESS &1 OF MEMORY TO BE
                                            TESTED
0100               ENDADH .     $0083
0100               COUNTR .     $0084  COUNTER AND SEED FOR TEST
0100               TMPADL .     $0085  WORKING ADDRESS POINTER
0100               TMPADH .     $0086
0100               TMPY   .     $0087  TEMPORARY STORAGE OF Y

                   KIM ROM ROUTINES USED

0100               CRLF   .     $1E2F  CARRIAGE RETURN - LINE FEED
0100               OUTCH  .     $1EA0  OUTPUT ASCII CHARACTER
0100               PRTBYT .     $1E3B  PRINT 1 HEX BYTE AS TWO ASCII
0100               OUTSP  .     $1E9E  OUTPUT BLANK


0100 20 2F 1E  INIT   JSR   CRLF   PRINT CR/LF
0103 A0 00     TEST   LDYIM $00    INITIALIZE INDEX REGISTER
0105 20 4A 01         JSR   SETEM  CREATE WORKING ADDRESS POINTER
0108 20 53 01  LOOPA  JSR   SUMUM  CALCULATE TEST DATA BYTE
0103 91 85            STAIY TMPADL STORE THE TEST BYTE
010D 20 5B 01         JSR   INCPTR INCREMENT THE WORKING POINTER
0110 D0 F6            BNE   LOOPA  MORE TO BE TESTED?
0112 20 4A 01         JSR   SETEM  REINITIALIZE WORKING POINTER
0115 20 53 01  LOOPB  JSR   SUMUM  RECALCULATE THE TEST DATA BYTE
0118 51 85            EORIY TMPADL CHECK MEMORY WITH CALCULATED
                                        TEST BYTE
011A D0 0F            BNE   ERROR  GO TELL IF TEST FAILED
011C 20 5B 01  RETURN JSR   INCPTR INCREMENT THE WORKING POINTER
011F D0 F4            BNE   LOOPB  MORE TO BE TESTED?
0121 A9 2B     ONCE   LDAIM '&     PRINT A "PLUS" TO INDICATE
                                        SUCCESS
0123 20 A0 1E         JSR   OUTCH  PRINT ASCII
0126 E6 84            INC   COUNTR SET UP NEW PATTERN
0128 4C 03 01         JMP   TEST   TEST UNTIL MANUAL RESET
012B 84 87     ERROR  STY   TMPY   SAVE Y
012D 48               PHA          SAVE THE BAD BIT PATTERN
012E 20 2F 1E         JSR   CRLF   PRINT CR/LF
0131 A5 86            LDA   TMPADH GET HIGH ADDRESS OF ERROR
0133 20 3B 1E         JSR   PRTBYT PRINT IT
0136 A5 85            LDA   TMPADL GET LOW ADDRESS OF ERROR
0138 20 3B 1E         JSR   PRTBYT PRINT IT
013B 20 9E 1E         JSR   OUTSP  PRINT A SPACE
013E 68               PLA          RESTORE THE BAD BIT PATTERN
013F 20 3B 1E         JSR   PRTBYT PRINT IT
0142 20 2F 1E         JSR   CRLF   PRINT A CR/LF
0145 A4 87            LDY   TMPY   RESTORE Y
0147 4C 1C 01         JMP   RETURN CONTINUE WITH THE TEST

                SUBROUTINES

014A A5 80     SETEM  LDA   BGNADL GET BEGINNING ADL
014C 85 85            STA   TMPADL MAKE A COPY
014E A5 81            LDA   BGNADH GET BEGINNING ADH
0150 85 86            STA   TMPADH MAKE A COPY
0152 60               RTS

0153 18        SUMUM  CLC          GET READY TO ADD
0154 A5 86            LDA   TMPADH GET WORKING POINTER ADH
0156 65 85            ADC   TMPADL ADD IN WORKING POINTER ADL
0158 65 84            ADC   COUNTR ADD IN COUNTER
015A 60               RTS   RETURN WITH CALCULATED TEST DATA BYTE
                                        IN A REGISTER

015B E6 85     INCPTR INC   TMPADL INCREMENT WORK POINTER ADL
015D D0 02            BNE   CMPADL PAGE NOT CROSSED
015F E6 86            INC   TMPADH INCREMENT WORK POINTER ADH
0161 A5 85     CMPADL LDA   TMPADL GET ADL OF WORK POINTER
0163 C5 82            CMP   ENDADL SEE IF END OF MEMORY TO BE
                                        TESTED
0165 D0 04            BNE   RTN    RETURN IF NO MATCH
0167 A5 86            LDA   TMPADH GET ADH OF END OF MEMORY TO BE
                                        TESTED
0169 C5 83            CMP   ENDADH SEE IF ADH'S MATCH
016B 60        RTN    RTS   RETURN WITH RESULTS OF CMP IN Z FLAG
```

The MICRO Software Catalogue: XV

Mike Rowe
P.O. Box 6502
Chelmsford, MA 01824

Name: **Mother Goose Rhymes**
System: **APPLE II**
Memory: **16K**
Language: **Integer BASIC and Machine Language**

Description: Children who love Mother Goose Rhymes will have fun with this interactive program using missing words. The program enjoyably guides children towards reading mastery.

Copies: **Just Released**
Price: **$9.95** for cassette
Includes: Cassette and loading instructions
Author: **George Earl**
Available from:
George Earl
1302 S. Gen. McMullen Dr.
San Antonio, TX 78237

Name: **SYM/KIM Appendix**
System: **SYM-1**
Memory: **1K**
Monitor Version:
1.0 or 1.1 — works with both
Language: **Machine Language**
Hardware: **SYM-1 alone,**no additions or expansion memory required

Description: This appendix is used as a supplement to the "First Book of Kim" (pub. by Hayden Books). It takes the entire recreational program section of the FBOK and provides the user with detailed changes to each program to allow them to run on an unmodified 1K SYM-1. The user is assumed to have access to the FBOK since only the changes are detailed in the appendix (along with explanations as needed). The basic goal of the appendix was to allow the purchaser of the most basic (1K) SYM to have some beginning software. Since the instructions indicate 'load the KIM program, modify parts as follows... then run', one might consider purchasing KIM games tapes and loading them using the KIM format load available on the SYM-1. Then he could modify the program and redump it for his own personal use later, using the SYM format. The modification techniques used in the appendix can also be used to convert other KIM programs for use on the SYM-1.

Copies: **20** delivered (as of 10/79) more available
Price: **$4.25,**First Class postpaid — Appendix only
$9.00, First Book of Kim, separately
$12.50, combo First Book of Kim and Appendix (FBOK and combo delivered 4th class or add $2.00 for first class. Cal. residents add 6% sales tax.
Available from Author:
Robert A. Peck
P.O. Box 2231
Sunnyvale, CA 94087

Name: **PET Quick Reference Card**
System: **PET**
Memory: **4K, 8K, 16K, and 32K**
Language: **English**
Hardware: **None**

Description: A complete summary of the Commodore PET BASIC language along with examples and definitions of every command. Also on the card is a table of the PET's graphic characters with their hexadecimal equivalents. Machine language programmers will find a table of important memory locations (for all model PETs), as well as information on the user port, PET sound, and the IEEE—488 interface bus. The information that PET owners used to have to hunt for in several books and magazines is now in one quick, convenient place!

Copies: **Just released**
Price: **$3.50** postpaid
Available from: Leading Edge Computer Products
P.O. Box 3872
Torrance, CA 90510

Name: **Dakin5 Programming Aids**
System: **APPLE II**
Memory: **48K**
Language: **Assembler/Applesoft II**
Hardware: **APPLE II, 2 Disk II's, and printer**

Description: Set of seven programs: 1) Lister — prints BASIC programs using full line capacity of printer. Peeker — displays or prints all or selected records from a text file. 3) Cruncher — removes REM statements and compresses code in Applesoft programs. 4) Text File Copy — copies a particular test file from one diskette to another. 5) Prompter — data entry subroutine that handles both string and numeric data. Options for using commas, decimal points, and leading zeros, with right-justified numerics. Alphanumeric data is left-justified with trailing spaces added as required. Maximum field length can be specified to prevent overflow in both numeric and alphanumeric fields. 6) Calculator — an addition/subtraction subroutine that handles numeric string data. Written in Assembler code, and using twenty place accuracy, it functions 40 times faster than if written in an equivalent BASIC subroutine. 7) Diskette Copy — formats an output disk, copies each track,and verifies that the output matches the input.

Copies: **Just released**
Price: **$39.95**
Includes: 35 page documentation and program diskette
Author: **Dakin5 Corporation** (developer of The Controller for Apple Computer, Inc.)
Available from: Local Apple dealers

Name: **Stock Market Option Account**
System: **APPLE II Computer**
Memory: **32K with Applesoft ROM**
48K with Applesoft RAM
Language: **Applesoft II**
Hardware: **Disk II, 132 column printer**

Description: The Stock Market Option Account program stores and retrieves virtually every option traded on all option exchanges. A self-prompting program allowing the user to enter short/long contracts. Computes gross and net profits/losses, and maintains a running cash balance. Takes into account any amending of cash balances such as new deposits and/or withdrawals from the account. Instantaneous read-outs (CRT or printer) of options on file, cash balances, P/L statement. Includes color bar graphs depicting cumulative and individual transactions. Also includes routine to proofread contracts before filing.

Copies: **Just Released**
Price: **$19.95 + $2.00 (P&H)** — Check or Money Order
Includes: Diskette and Complete Documentation
Available from:
Mind Machine, Inc.
31 Woodhollow Lane
Huntington, N.Y. 11743

Name: **IFO-DATA BASE MANAGER PROGRAM**
System: **APPLE II OR APPLE PLUS COMPUTERS**
Memory: **48K**
Language: **APPLESOFT II** on Firmware (or APPLE II plus computer)
Hardware: **Single Disk Drive and Serial or Parallel Printer**

Description: The IFO (Information File Organizer) Program can be used for sales activity, inventory, check registers, balance sheets, price markups, library functions, client/patient billing and many more applications. In order to use the IFO no prior programming knowledge is required. All commands are in English and are self-prompting. Up to 20 header can be created and a maximum of 1000 records can be stored on a single diskette. Information can be sorted (ascending or descending order) on any field and cross-referenced using 5 criteria on up to 3 levels of searches. Mathematical functions (adding, dividing, multiplying, squaring) can be performed on any 2 columns of data or on 1 column of data in combination with a constant to create a new column of data. Information in the data base can be printed in up to 10 different report formats using a 40, 80 or 132 column, serial or parallel printer or may be viewed on the screen only. There are numerous error protection devices in the program so that the program is easy to use and allows the user to run the program error free.

Copies: **Just Released.**
Includes: Program Diskette and Instruction Manual
Price: **$100 (**Manual Only:**$20)**
Author: **Gary E. Haffer**
Available From:
Software Technology for Computers
P.O. Box 428
Belmont, MA 02178

Name: **BASIC Programmer's Toolkit**
System: **PET**
Memory: **All**
Language: **Machine Language Firmware**
Hardware: **All standard PETs, or with Betsi, Expandamem or Skyles add-on memory**

Description: The BASIC Programmer's Toolkit is a collection of programming aids, coded in 6502 machine language, and delivered as a 2KByte add-on ROM. Adds 10 new commands to the PET; namely, AUTO, RENUMBER, DELETE, HELP, TRACE, STEP, OFF, APPEND, DUMP and FIND. Commands are entered as shown above, with optional parameters. Guaranteed to make the developing and debugging of BASIC programs for the PET faster and easier.

Copies: **Several thousand** in use already
Price **$49.95 or $79.95** (depending on version)
Author: **Palo Alto IC's,** a division of Nestar Systems,Inc.
430 Sherman Avenue
Palo Alto, California 94306
Available from: Local PET dealers

Name: **Astronomer**
System: **APPLE II**
Memory: **16K with Applesoft ROM, 32K with Applesoft RAM**
Language: **Applesoft II**
Hardware: **Applesoft ROM (optional)**

Description: Astronomer applies the personal computer to aspects of astronomy which previously were available only in almanacs for specific times and conditions. Using expressions in the Almanac for Computers (U.S. Naval Observatory), times of sunrise-sunset-twilight, sidereal time, precession and Julian Date are calculated in this program for any date, time or location. The computations are completed without delay and conditions are set through an efficient user-interface.

Copies: **New Program**
Price: **$10 + $2** handling and postage
Includes: Complete documentation
Author: **Bruce Bohannon**
Available from:
Bruce Bohannon
2212 Pine Street
Boulder, CO 80302

Name: **DISCOUNT & YIELD**
System: **PET**
Memory: **8K**
Language: **BASIC**
Hardware: **PET(8K) With Cassette**

Description: Discount and Yield is designed to provide the time-value calculations necessary to determine the required discount or yield when purchasing or selling contract for deeds, land contracts or mortgages. The program will also handle the complexity of calculating discounts and yields when prepayments are made at nonscheduled intervals.

Copies: **Just Released**
Price: **$8.95**
Includes:
Cassette and instructions
Author: **D.J. Romain**
Available from:
D. J. Romain, P.E.
405 Reflection Road
Apple Valley, MN 55124

6502 Bibliography: Part XV

William R. Dial
438 Roslyn Avenue
Akron, OH 44320

505. MICRO No. 13

Dial, Wm. R., "6502 Information Resources Updated", pgs. 29-30.
Additional and updated information on the publisher's address, subscription rates etc. for the publications cited in the 6502 Bibliography.

Lipson, Neil D., "The Color Gun for the Apple II", pgs. 31-32
Turn your Apple into a device which will determine the colors of any object.

Tripp, Robert M., "Ask the Doctor--Part V", pgs. 34-36
Discussion of AIM or SYM problems in loading KIM format cassette tapes, a short routine to get by the SYM "2F" loading bug and a routine which mimics the KIM SCANDS routine on the SYM.

Reich, L.S., "Computer-Determined Parameters for Free-Radical Polymerization.", pgs 38-40
Program for determining parameters for weight-fraction versus polymer size. Includes Example run using polystyrene data.

DeJong, Marvin L., "AIM 6522 Based Frequency Counter", pgs. 41-42
Usingthe AIM 65 as a six digit frequency counter capable of counting to at least 450kHz.

Scarpelli, Anthony T., "KIM—The Tunesmith", pgs. 43-52
Play, compose, save and play back music on your KIM.

Rowe, Mike (staff), "The MICRO Software Catalog:IX" pgs. 53-54
Ten interesting software offerings are reviewed.

Gieryic, Jack, "SYM-1: Speak to Me", pgs. 57-58
Some starting techniques for storing speech. Lots of memory is the key——about 5K per second of speech.

Kemp, David P., "Reading PET Cassettes without a PET", pgs. 61-63
A program is given which makes it possible for a SYM-1 to read a PET cassette.

506. Recreational Computing 7, No. 6 (May/June 1979)

Day, Jim, "PT2: Apple Scan Simulation", pg. 5.
An Applesoft II program that simulates a high resolution PPI scan.

507. The Cider Press 2 No. 3 (June 1979)

Larsen, LeRay, "Having Disk Problems?"pg. 5
A bad sector of a disk can often be rectified by putting a small amount of recording tape lubricant on the window. Then erase and reinitialize.

Wilson, Gene, "Apple II Utility Disk Software Review", pg. 5
Review of a diskette by Roger Wagner of Southwestern Data Systems, P.O. Box 582, Santee, CA 92071

Anon, "Disk of the Month — June, 1979", pg. 4
19 programs totaling some 60 kilobytes.

508. Byte 4 No. 6 (June, 1979)

Watson, Alan III, "More Colors for your Apple", pgs. 60-68
How to get additional High Resolution Colors out of your Apple.

Leedom, Bob, "Approximation Makes Magniture of Difference", pgs. 188-189 (June, 1979)
Some tips in adapting a fast Fourier transform program for the 6800 to a KIM 6502 system.

509. Kilobaud Microcomputing No 31 (July, 1979)

Lindsay, Len, "PET-Pourri", pgs. 6-7
Information on the new 32K PETs with full size keyboards, how to modify programs for the new PET, further discussion of cassette problems, etc.

Anon, "Ohio Scientific Small Systems Journal", pgs. 8-11
Discussion of the OS-DMS data management system.

Pepper, Clement S., "Safe Ports", pgs. 60-62
Protect your I/O ports with this bidirectional buffer. Implemented on a KIM-1.

Chamberlain, Bruce S., "OSI's Superboard II;;, pgs. 66-70
A favorable review of this inexpensive micro board.

Lindsay Len, "Teach an old PET New Tricks" pgs. 72-74
Some reference charts to make less difficult the job of modifying programs for the OLD PET to run on the NEW PET

Sybex, 2020 Milvia St., Berkeley, CA 94704, pg. 104
Rodney Zak's new book "6502 Applications Book" is advertised.

Hallen, Rod, "The 6502 and Its Little Brothers" pgs. 124-126
A discussion of some of the other members of the 65xx family.

510. 6502 User Notes No. 15 (June, 1979)

Williams, J.C., "A 32K Dynamic RAM Board for the KIM-4 Bus" pg. 1
Constructional Article.

Green, Jim, "650X Save and Restore Routines pg. 4
Routines save and recover A,Y, and X register values.

Kantrowitz, Mark, "Telephone Dailer" pgs. 6-9
Saves and dials up to 16 different telephone numbers.

Flynn, Christopher, "Some Important BASIC Mods" pg. 9
MLDSPT can be used to activate user-written machine language routines. ARRSAV/ARRLOD provides an easy way to save and load data on cassette from BASIC arrays.

Mulder, Bernhard, "Focal Mods" pg. 13
Speed it up a little with these mods.

Clements, William D., Jr., "Tiny BASIC Cassette Save and Load"pg. 13-14
Add save and load commands to your TINY BASIC.

Day, Michael E., "TINY BASIC Strings" pgs. 14-16
Here is a string MOD accessed thru USR

Fatovic, J., "Assembler" pgs. 16-17
A symbol table sort for the MOS/Aresco Assembler.

Scanlon, Leo, "Warning" pg. 18
A warning about the types of thermal paper to use in the AIM 65. Apparently some types are abrasive and can ruin the printer head.

Goga, Larry, "Notes on AIM User I/O" pgs. 18-20
All about RDRUB and also a Memory test Program.

Campbell, John R., "Modification to KIMSI to add 4K to RAM to Memory Space Below Monitor" pg. 20
How to add 4K from address $0000 to $13FF.

Schilling, Heinz J., "CPU Bug" pg. 22
A bug in the JMP Indirect instruction of the 6502.

The Editor, "6522 Info and Data Sheet Corrections" pg. 22
A number of corrections are given.

Lewart, Cass, "Extending the Range of KIM-1 Timer to 1:32640" pgs. 22-23
A simple fix to make the extension.

DeJong, Marvin L., "SYM and AIM Timer Locations." pg. 23
This will help in modifying programs to run on AIM or SYM.

Boisvert, Conrad, "Use of the RDY Line to Halt the Processor" pg. 23.
A simple circuit is given.

Nazarian, Bruce, "Additions to the MTU Software Package" (KIM) pg. 26
Some additions and changes for Hal Chamberlain's DAC Software.

Lewart,Cass R., "A Simple Microprocessor Interface Circuit" pg. 26
An interface to let KIM control LEDs, relays, or AC operated appliances.

511. Personal Computing 3 No. 7 (July, 1979)

McKee, Paul, "Merging on the Challenger", pg. 8
Discussion of merging two BASIC programs.

Franklin, Larry, "Line Renumbering on the OSI" pg. 9
Discussion and modification of a line renumbering program.

Scarpelli, Anthony T., "Making Music with Fractals" pgs. 17-27
Random Tones on the KIM-1.

512. Southeastern Software Newsletter, Issue 10 (June, 1979)

Banks, Guil, "Diskette Space", pgs. 1-2
Machine Language program to tell how much space is left on a diskette. Also an Integer Basic program to call up the routine. With tutorial discussion by the editor.

Anon, "Input/Output Control Block", pg. 3
Discussion of uses for the IOB and Device Characteristics Table for the Apple II DOS 3.2 System.

Howard, Clifton M., "How to Use the TOKEN Routine", pg. 4
A step-by-step description of how to use the TOKEN Routine.

Anon, "Shorthand Commands for 3.2", pg. 5
How to add a series of shorthand controls to the Apple DOS 3.2 system.

Anon, "Turning Your Printer On", pg. 6
Short program to turn printer on and off.

513. Stems from Apple 2 Issue 6 (June, 1979)

Griffith, Joe, "Plotting Algebraic Equations", pg. 3
Several programs for different types of equations.

Hoggatt, Ken, "Ken's Korner", pgs. 6-7
Discussion of the Apple Contributed programs Nos. 3, 4 and 5. Also covered are the character generator and the character table.

Anon, "Apple Stem's Software List"
A list of 100 programs for the Apple was enclosed with the newsletter.

514. Call — Apple 2, No. 5 (June, 1979)

Golding, Val J., "Hiding Out in BASIC", pg. 5
Discussion of methods of imbedding machine code in Basic, Poke Statements, Monitor Routine, Data and Read Statements, Linker, and other routines.

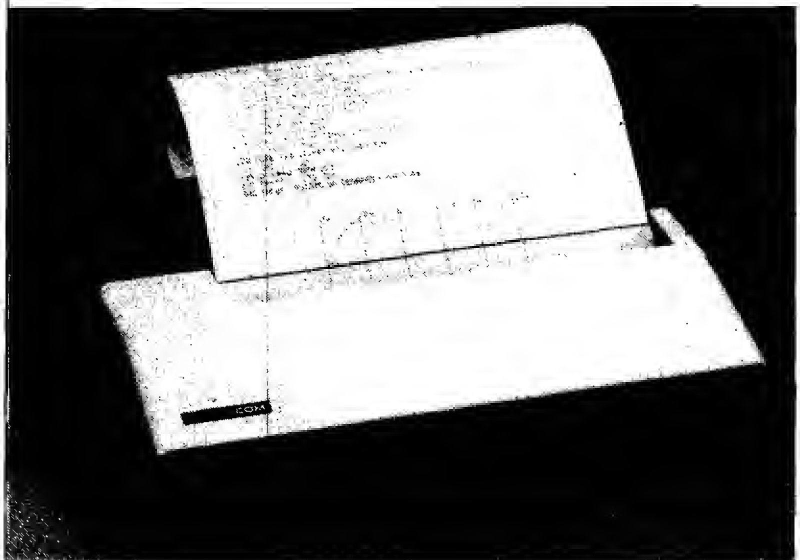
Winston, Alan B., "The Multilingual Apple", pgs. 11-13
Discussion of the Fourth Language and a look at the CHRs pseudo-function and GET C$ for Apple Integer Basic.

Anon, "DOS 3.2 Changes", pg. 15
Rewriting file-oriented programs to accommodate the change to the Apple DOS 3.2 System.

Thyng, Mike, "Applemash", pg. 5
How to pass basic serial data thru your Apple Communications Card.

Kotinoff, Jeff, "LORES Color Picture", pg. 19
Two color programs for the Apple II.

Skyles Electric Works — PAL-80™



- 80 characters per line
- 8½ inch wide thermal paper
- Full graphics at 60 dots/inch
- Interfaced to PET
- Works with all PET peripherals
- 40 character per second rate
- Microprocessor controlled
- Bidirectional look-ahead printing
- Quiet operation
- No external power supplies
- Only two driven parts
- High reliability
- Clear 5 x 7 characters
- Attractive metal and plastic case

The Skyles PAL-80™ is a high speed thermal printer **offering the combination of text printing at 80 characters per line and continuous graphics at 60 dots per inch.** In the text mode, upper and lower case data are printed at 40 characters per second. The 5 x 7 characters provide clear readable copy on white paper; no hard to find, hard to read aluminized paper.

In the graphics mode, seven bits of each byte correspond to the seven dots in each of the 480 print positions per line. Since the computer driving the printer has full control over every print position, it can print graphs, bar charts, line drawings, even special and foreign language symbols. Despite its low cost, the Skyles PAL-80 is a true intelligent printer with full line buffering and bi-directional look-ahead printing.

High reliability is designed in: The thick film thermal print head has a life expectancy of 100,000,000 characters. Two DC stepping motors provide positive control of the print head and the paper drive.

The Skyles PAL-80 operates directly from a 115V 60 Hz line (230V 50 Hz available). No external power supplies are required.

It comes complete with an interface for the PET: a two and a half foot cable plugs into the IEEE interface at the back of your PET. Works with all PET models and PET or Skyles peripherals.

Please send me________ **Skyles PAL-80 printer(s)** complete with 2½ foot interface cable to attach to my PET at **$675.00 each*** (Plus $10.00 shipping and handling). I also will receive a test and graphics demonstration tape at no additional charge and over 150 feet of 8½ inch wide black on white thermal paper $________

I would also like to order _______ rolls of 8½ inch wide by 85 ft. long thermal paper (black ink) at $5.00 each $ ____________

________ 10 roll cartons at $45.00 $____________

VISA, Mastercharge orders call **(800) 227-8398**
California orders call **(415) 494-1210**

*California residents add 6 to 6½% sales tax where applicable.

PAL-80 SPECIFICATIONS

TEXT	
Format	80 characters per eight inch line 6 lines per inch nominal
Print speed	40 characters per second
Line Feed	50 milliseconds nominal
Character Set	96 Characters, including upper and lower case, numerals, and symbols
GRAPHICS	
Format	480 print positions per line
Print Speed	240 print positions per second
COMMON	
Paper	8½ inch wide thermal paper, available in 85 foot folls, black image on white
Dimensions	12"W x 10"D x 2¾"H
Weight	8 lbs (3.6 kg)

Skyles Electric Works | 10301 Stonydale Drive
Cupertino, CA 95014 • (408) 735-7891